FTD-TT- 65-942

AD633642
TT66-61394

TRANSLATION

INTRODUCTION TO CYBERNETICS

By

V. M. Glushkov

FOREIGN TECHNOLOGY DIVISION

AIR FORCE SYSTEMS COMMAND

WRIGHT-PATTERSON AIR FORCE BASE

OHIO

FOREIGN TECHNOLOGY DIVISION

FTD-TT- 65-942/1+2

UNEDITED ROUGH DRAFT TRANSLATION

INTRODUCTION TO CYBERNETICS

BY: V. M. Glushkov

English pages: 469

TM5000976

THIS TRANSLATION IS A RENDITION OF THE ORIGINAL FOREIGN TEXT WITHOUT ANY ANALYTICAL OR EDITORIAL COMMENT. STATEMENTS OR THEORIES ADVOCATED OR IMPLIED ARE THOSE OF THE SOURCE AND DO NOT NECESSARILY REFLECT THE POSITION OR OPINION OF THE FOREIGN TECHNOLOGY DIVISION.	PREPARED BY: TRANSLATION DIVISION FOREIGN TECHNOLOGY DIVISION WP-AFB, OHIO.

FTD-TT- 65-942/1+2

AFLC-WPAFB-MAR 66 72

Date 17 Feb. 19 66

S

+2

Akademiya Nauk Ukrainskoy SSR

Nauchnyy Sovet Po Kibernetike

V. M. Glushkov

VVEDENIYE V KIBERNETIKU

Izdatel'stvo Akademiya Nauk Ukrainskoy SSR

Kiev - 1964

324 pages

FTD-TT-65-942/1+2

TABLE OF CONTENTS

Foreword. 2

Chapter 1. Abstract Theory of Automata 9

§1. Alphabetic Operators and Algorithms 9
§2. Normal Algorithms. 25
§3. The Kolmogorov-Uspenskiy Algorithmic Diagram 40
§4. Other Theoretical Algorithmic Systems. 49
§5. The Concept of Algorithmically Insoluble Problems. . 57

Chapter 2. Boolean Functions and Propositional Calculus . . . 68

§1. Concept of Boolean Functions 68
§2. Boolean Algebra. 81
§3. The Concept of Complete Sets of Boolean Operations . 99
§4. Application of Boolean Algebra in the Theory of Combination Circuits 110
§5. The Concept of Propositional Calculus. 127

Chapter 3. Theory of Automata 145

§1. Abstract Automata and Automation Representations . . 145
§2. Events and Representation of Events in Automata . . 158
§3. Analysis of Finite Automata 166
§4. Abstract Synthesis of Finite Automata 173
§5. Minimization of Abstract Automata 183
§6. Structural Synthesis of Finite Automata. 192

Chapter 4. Self-Organizing Systems. 206

§1. Concept of Self-Alteration and Self-Organization in Automata. 206
§2. Some Auxiliary Information From Probability Theory . 217
§3. A Quantitative Measure of Self-Organization and Self-Improvement in Automata 231
§4. Automata With Random Transitions 241
§5. The Problem of Pattern Recognition Training. 254
§6. Theory of Training of Discrete α-Perceptrons 272
§7. Operation of the Discrete α-Perceptrons in the Self-Learning Regime 291
§8. Logical Classification Systems and Conditional Probability Machines 303
§9. Self-Organization and Self-Adaptation. Methods of Solution of Complex Variational Problems 322

Chapter 5. Electronic Digital Machines and Programming . . . 341

1. The Universal Program Automaton 341

§2. Structure of the Modern Universal Program Automata . 353
§3. The Concept of Programming 368
§4. The Universal Algorithmic Language Algol-60 382
§5. Examples of Programming Using Algol-60 405

Chapter 6. Predicate Calculus and the Problem of Automation of the Scientific Creative Processes 421

§1. Basic Concepts of Predicate Calculus 421
§2. Formal Arithmetic and the Godel Theorem 440
§3. Concept of Automation of Proofs and Construction of Deductive Theories 453

References. 470

This book contains the collected and unified material necessary for the presentation of such branches of modern cybernetics as the theory of electronic digital computers, theory of discrete automata, theory of discrete self-organizing systems, automation of thought processes, theory of image recognition, etc. Discussions are given of the fundamentals of the theory of boolean functions, algorithm theory, principles of the design of electronic digital computers and universal algorithmical languages, fundamentals of perceptron theory, some theoretical questions of the theory of self-organizing systems.

Many fundamental results in mathematical logic and algorithm theory are presented in summary form, without detailed proofs, and in some cases without any proof.

The book is intended for a broad audience of mathematicians and scientists of many specialties who wish to acquaint themselves with the problems of modern cybernetics.

FOREWORD

The objective of the present book is to acquaint the reader with several new scientific directions which constitute the basis of cybernetics in its modern concept. In the most general framework all these trends can be subdivided into two major groups – the general theory of information conversion, and the theory and principles of the design of various kinds of information converters. However, the material which can be associated with these major trends is so extensive that it could hardly be presented even in summary form in a single book. Therefore it has been necessary to make a selection of the material in accordance with some general principles.

The material for the present book has been selected in accordance with two basic principles. The first principle is the requirement for a sufficiently rigorous formulation of the material to permit presenting it in the form of a mathematic theory (although with the bent in the direction of practical simulation which is characteristic of cybernetics). The second principle is that the author limits himself, as a rule, to the digital methods of representing information and the digital conversion of information.

As a result of the selection, the book contains the following basic sections: algorithm theory (including programming for general purpose electronic digital computers and universal algorithmic languages for programming), theory of discrete automata (including the theory of boolean functions and the concept of the principles of the design of general-purpose electronic digital computers), theory of discrete self-organizing systems (including elements of the theory of optimal decisions) and, finally, mathematical logic (propositional calculus, restricted predicate calculus and formal arithmetic), considered as a basis for the automation of the process of the design of design of deductive (based on a particular system of axioms) theories.

The degree of detail of the presentation of the material is determined first of all by the degree of its novelty. The newer branches, related to cybernetics itself, are discussed in greater detail, the fundamental theorems are supplied with quite detailed proofs. At the same time, in such branches as abstract algorithm theory and mathematical logic, which have developed within the framework of traditional mathematics, the material is presented more briefly, proofs, as a rule, are omitted.

The author has attempted, however, to give an understanding of the basic ideas and methods which are used to establish the validity of such fundamental, from the point of view of mathematic logics, propositions as the Godel theorem on the incompleteness of arithmetic or the theorems which establish the algorithmic insolubility of particular problems.

The book does not pretend to replace specialized monographs on the individual sections which are included here. Its primary intention is to aid a wide audience of mathematicians and engineers to master that minimum of knowledge which is necessary for work in the field of the theoretical problems of modern "digital" cybernetics. It is well known that the existence of detailed monographs on a particular theme does not always make it possible for readers without specialized preparation to become acquainted with the subject. Convincing proof of this is the fact that in spite of the existence of specialized monographs, such a theorem as that of Godel mentioned above, which is of fundamental importance for all of mathematics, remains unknown to large numbers of mathematicians except for hearsay.

As for the present book, it presents to the reader (but only in one chapter, the fourth), the knowledge of only those elements of mathematical analysis and probability theory which are known to practically every engineer, without mentioning mathematicians. The less widely known mathematical results necessary for the understanding of the main content of the book are included as supplementary material. An example of this sort of supplementary material might be the series of propositions of probability theory presented in Chapter 4, §2.

In case the reader wishes to extend his knowledge in a particular area or become acquainted with the detailed proofs of those propositions which, although included in the book, are not proved in detail, we shall make a summary of the contents of the book with an indication of the specialized monographs (in Russian) pertaining to the individual sections. Unfortunately, this sort of monograph cannot be found pertaining to all the sections of the book.

The first chapter presents a description of the basic theoretical universal algorithmic systems (normal Markov algorithms, the Kolmogorov-Uspenskiy algorithmic system, recursive functions, the Post algorithms, and the Turing machine). Also presented are the basic principles of the proofs of the algorithmic insolubility of certain very simple mass problems.

At the present time there is no unifying monograph available on the intire theory of algorithms as a whole. Moreover, not all the questions mentioned above are covered in any detail in the monographic literature. Among the principal monographs on the individual algorithmic systems we might mention the following: on the theory of normal algorithms, Theory of Algorithms, A.A. Markov (Ref 53); on the theory of recursive functions and Turing machines, Introduction to Metamathematics, S.C. Kleene (Ref 42) and Course on Computable Functions, V.A. Uspenskiy (Ref 76).

The theory of boolean functions and its applications to the theory of discrete automata circuits are presented in the second chapter. These questions are discussed in greater detail in the monograph of V.M. Glushkov, Synthesis of Digital Automata (Ref 26).

In addition, the second chapter covers the fundamentals of propositional theory. More detail on propositional calculus can be found, for example, in the monograph of P.S. Novikova, Elements of Mathematical Logic (Ref 61).

The third chapter is devoted to the abstract and structural theory of discrete (finite) automata. The questions relating to this subject are considered in more detail in the monograph of Glushkov mentioned above. These questions are covered from somewhat different positions in the monograph of N.Ye. Kobrinskiy and V.A. Trakhtenbrot, Introduction to the Theory of Finite Automata (Ref 47).

The fundamentals of the theory of discrete self-organizing systems are presented in the fourth chapter. A definition is given of the quantitative measure of self-organization and self-learning, a study is made of the behaviour of random automata and automata operating in conditions of random external inputs. Special attention is devoted to the problem of the recognition of images and the theory of one class of devices (the so-called α-perceptron) intended for the resolution of this problem. Some questions of the simulation of conditioned reflexes are considered, and also questions of the teaching of meaning recognition and the generation of new concepts. At the end of the chapter, in connection with the idea of self-adjustment and extremal regulation, descrptions are given of several general methods for the solution of extremal problems (the method of steepest descent and its refinement, the simplex method of solution of the problems of linear programming and the so-called method of sequential analysis of variants for the solution of the problems of dynamic programming).

So far no unifying monograph is avialable on the material of the fourth chapter. Moreover, almost all the questions discussed in this chapter (with the exception of the method of steepest descent and the simplex method) have not yet been covered in the monographic literature. Several questions allied with those considered in this chapter (but not completely identical to them) are covered in Neurodynamics, F. Rosenblatt which has not yet been translated into Russian. A large number of monographs is devoted to the methods of solution of experimental problems (with the exception of the method of sequential analysis of variants). However, we shall not list them here since these questions have no direct relation to the primary theme of the present book.

The fifth chapter covers the basic principles of the design of the general-purpose electronic digital computers and the programming for these machines. So many monographs have been devoted to this question that it would be very difficult to list them all. In particular, we might cite on the subject of programming the monograph of B.V. Gnedenko, V.S. Korolyuk and Ye.L. Yushchenko, Elements of Programming (Ref 31). As for the principles of computer design, in spite of the existence of many good specialized monographs on this question, a detailed presentation of the material in the framework we need does not exist; the principles of the design of the electronic digital computers are presented, as a rule, in isolation from the general theory of algorithms.

In addition, the fifth chapter presents a detailed description of the universal algorithmic language ALGOL-60 and gives examples of ALGOL programming of various problems, primarily from the theory of self-organizing systems. In particular, a discussion is given of the question of the programming of the perceptron learning process and of a simplified model of the process of biological evolution. Again, on this question there is little information in the monographic literature: Report on the Algorithmic Language ALGOL-60 (edited by P. Naur), published by the Computer Center of the USSR Academy of Sciences (Moscow, 1960) is of a reference nature and not suitable for paractical instruction on the ALGOL language.

In the last (sixth) chapter there is given a summary exposition of the fundamentals of the restricted predicate calculus (including the formal system of Gentzen) and of formal arithmetic (including the Godel theory on arithmetic incompleteness). Detailed proofs of the propositions presented can be found in the previously cited monographs of Kleene and Novikov. This chapter also contains elements of the automation of proofs and formulations of theorems in deductive theories. The questions touched on here have not yet been covered in the monographic literature.

As indicated by the list of the material presented in the book, several interesting branches of modern cybernetics are not included in the book. Considering the criteria mentioned previously for the selection of material, we could, for example, include a presentation of the fundamentals of mathematical linguistics or elements of game theory. However, even without this, the considerable size of the book has forced the author to refrain from attempts to include any additional material. At the same time, the contents of the book do encompass those questions which at the present time as usually considered the basis of theoretical cybernetics (with account for limiting ourselves to discrete methods). The author hopes, therefore, that the book will be of assistance in mastering the mathematical apparatus of cybernetics and preparing for work in the theoretical fields to individuals occupied in individual applied aspects of cybernetics and also to the individuals interested in the theoretical problems of cybernetics.

In the present book extensive use has been made of material from courses on the various branches of cybernetics and mathematical logic presented b y the author at Kiev University and at the Kiev House of Scientific and Technical Propaganda in 1959-1962. A part of this material (theory of algorithms, for example) has been published previously for service use. The present book can be considered to be the first sufficiently complete textbook for students of the branches of cybernetics mentioned above.

Chapter 1

ABSTRACT THEORY OF AUTOMATA

§1. ALPHABETIC OPERATORS AND ALGORITHMS

In modern mathematics it is customary to call the structurally specified correspondences between words in abstract alphabets algorithms.

Any finite ensemble of objects, termed the letters of a given alphabet, is called an abstract alphabet. The nature of these objects is a matter of complete indifference to us. For example, the letters of the alphabet of any language (Russian, Latin, Greek, etc.), digits, any symbols, figures, etc., can be considered to be letters of abstract alphabets. If we wish to, we can introduce an abstract alphabet whose letters will be considered to be entire words of any particular language (Russian, for example). It is important only that the alphabet considered be finite, i.e. that it consist of a finite number of letters.

Introducing the concept of an (abstract) alphabet, we define a word in this alphabet as any finite ordered sequence of letters. For example, in the alphabet $A = A(x,y)$ consisting of the two letters x and y we consider any sequence $x, y, xx, xy, yx, yy, xxx, \ldots$ to be words. The number of letters in a work is termed normally the length of this word, so that the words we just listed in the alphabet have respectively the lengths 1, 1, 2, 2, 2, 2, 3,...

Along with words of positive length (consisting of no less than one letter), in many cases it is convenient to consider also an empty word, not containing even one letter. In the present chapter use is made of the small Latin letter e to designate an empty word. Sometimes, however, it is convenient to designate the empty word in complete accordance with its definition, not writing any letter in the place corresponding to this word.

We note that, with the accepted definition, the concept of a word in the Russian alphabet will differ from the concept of a word as accepted in ordinary language. With our definition, words are to be considered any combination of letters, including meaningless combinations: the combinations of letters "algorithm", "mathematics", "'klt", "dddd" must to an equal degree be considered words of the Russian alphabet (considered as an abstract alphabet).

With expansion of an alphabet, i.e., with inclusion in its composition of new letters, the concept of the word may undergo significant changes. If, for example, we expand the Russian alphabet by the "letters" ("   " – parentheses) and (, – comma), then the four words which we have just written out in the Russian alphabet can be considered as a single word in the alphabet expanded in this fashion. By complementing the Russian alphabet with the punctuation marks and the separation mark (empty space left between two neighboring words), we can if we wish consider entire phrases, paragraphs and even entire books as individual words.

In just the same way, the expression 69 + 72, which is two words (69 and 72) in the alphabet A of the 10 digits (0,1,2,3,4,5,6,7,8,9), joined by the sum sign, can be considered as a single work in the expanded alphabet A which is obtained as the result of joining to it the new letter "+" (sum sign).

Alphabetic operator or alphabetic representation is the term given to any correspondence (function) which associates words in a particular alphabet to words in the same or another fixed alphabet. The first alphabet is here termed the input, and the second the output alphabet of the given operator. In the case of coincidence of the input and output alphabets, we say that the alphabetic operator is given in the corresponding alphabet.

Hereafter we consider primarily single-valued alphabetic operators, associating to each input word (word in the input alphabet of the operator) no more than one output word (word in the output alphabet of the operator). If the alphabetic operator does not associated with a given input word p any output word (including an empty word), then we say that it is not defined on this word. The ensemble of all words on which an alphabetic operator is defined is termed its domain of definition.

On the basis of the foregoing, in the future we shall always understand (if not otherwise specified) by the term "alphabetic operator" a unique, generally speaking, partially defined mapping of a set of words in the input alphabet of the operator into a set of words in its output alphabet.

Thanks to the possibility of specifying the alphabetic operators on less than all the words, we can, without loss of generality, every time consider that the input and output alphabets of the operator coincide. For this it is sufficient, clearly, to combine the input and output alphabets of the given operator φ into one common alphabet A and to consider the operator φ as an operator in this combined alphabet, specified only on those words which appeared in the primitive region of definition of the operator φ.

With each alphabetic operator there is associated an intuitive concept on its complexity. The simplest operators are those which perform letter-by-letter mapping. This mapping consists in each letter x

of the input word p being replaced by some letter y of the output alphabet operator, depending only on the letter x and not on the choice of the input word p. Letter-wise mapping is completely defined by specifying the correspondence between the letters of the input and output alphabets.

The so-called coding transformations, which for brevity we shall term simply codings, are of great importance for the later discussion. In the simplest case the words in one alphabet, say in alphabet A, are coded by words in the other alphabet, B, as follows: to each letter a_i of the alphabet A there is associated some finite sequence b_{i_1}, b_{i_2}, ...b_{i_k} of letters in the alphabet B, called the code of the corresponding letter, such that to the different letters of the alphabet A there are associated different codes.

For the construction of the desired coding transformation it is sufficient now to replace all the letters of any word p in the alphabet A by the codes corresponding to them. The word thus obtained in the alphabet B we term the code of the original word p. We stipulate that the coding transformation must necessarily be reversible. In other words, different words in alphabet A must have different codes. The condition of reversibility of the coding is nothing other than the condition of mutual uniqueness of the corresponding coding transformation.

It is easy to see that reversibility of the coding is not ensured by the single condition that the codes of the various letters (words of length 1) be different. Actually, if to the letter a_1 there is associated the code bb, and to the letter a_2 the code b, then the code bbb will clearly correspond both to the word a_1a_2 and to the words a_2a_1 and $a_2a_2a_2$.

It is not difficult to verify that the coding will be reversible whenever the following two conditions are fulfilled:

a) the codes of the different letters of the original alphabet A are different;

b) the code of any letter of the alphabet A cannot coincide with any of the initial segments of the codes of the other letters of this alphabet.*

Actually, let us assume that both of these conditions are satisfied and let the word $q = b_{i_1} b_{i_2} \ldots b_{i_n}$ be the code of some word $p = a_{j_1} a_{j_2} \ldots a_{j_m}$ in the alphabet A. Let us show that from the code $\underline{q}$ we can uniquely recover the word $\underline{p}$. In view of condition b) only one initial segment of the word $\underline{q}$ can coincide with the code of any letter of the alphabet A. It is clear that the code of the letter a_{j_1} is such a segment. Discarding this segment, we obtain the code q_1 of the word $p_1 = a_{j_2} \ldots a_{j_m}$. Applying to it the same reasoning, we restore uniquely the following letter (a_{j_2}) of the word $\underline{p}$, and so on. Using this technique, all the letters of the word $\underline{p}$ are restored one after the other. Consequently, to any given code there can correspond only one word in the alphabet A, which proves the reversibility (mutual uniqueness) of the coding transformation.

Condition b) is satisfied if the codes of all the letters of the original alphabet have identical length. By convention we call the coding in this case normal. Use of coding permits reducing the study of arbitrary alphabetic transformations to alphabetic transformations in some once-and-for-all selected standard alphabet. Most frequently, as such a standard alphabet there is chosen the so-called binary alphabet, consisting of two letters which are usually identified with the digits 0 and 1.

Let A be an arbitrary alphabet and B be a standard alphabet (binary, for example) consisting of more than one letter. If n is the number of letters in alphabet A and m is the number of letters in alphabet B, then we can always select the number k so as to satisfy the inequality

$$m^k > n. \qquad (1)$$

Since the number of different words of length k in the m-letter alphabet is clearly equal to m^k, then inequality (1) shows that we can code all the letters in alphabet A with words of length k in alphabet B so that the codes of the different letters are different. Any such coding will be normal and will generate, in light of what was said above, a reversible coding transformation of the words in alphabet A into words in alphabet B. We designate this transformation by α and use α^{-1} to designate the reverse transformation which transforms each word q in the alphabet B, which is the code of some word p in alphabet A, into the word p.

Now if φ is an arbitrary alphabetic operator in alphabet A, then the transformation $\psi = \alpha^{-1}\varphi\alpha$ obtained as the result of sequential performance of the transformations α^{-1}, φ and α will be, obviously, some alphabetic operator in the standard alphabet B. We term this operator the alphabetic operator in the alphabet B, conjugate (with the aid of the α coding) with the alphabetic operator φ.

The operator φ is uniquely recovered from the conjugate operator ψ and the corresponding coding transformation α

$$\varphi = \alpha\psi\alpha^{-1}. \qquad (2)$$

With the aid of this equation, and also its dual equation which was written previously

$$\psi = \alpha^{-1}\varphi\alpha \qquad (3)$$

the arbitrary alphabetic operators are reduced to alphabetic operators in the standard alphabet. This reduction, of course, can be performed by an infinite number of different methods, since there exist infinitely many different codings of words in any given alphabet by words in the standard alphabet.

The described reduction can also be accomplished in the case of alphabetic operators for which the input and output alphabets are different. For example, let φ be an arbitrary alphabetic operator with the input alphabet A and the output alphabet C, let B be the standard alphabet, let α be any (reversible) coding of words in the alphabet A by words in the standard alphabet, and let γ be an analogous coding of the words in alphabet C.

Now it is easy to see that the transformation $\psi = \alpha^{-1}\varphi\gamma$ is an alphabetic operator in the standard alphabet B by which under the condition of knowing the coding transformations α and γ the original transformation φ is uniquely restored.

The concept of the alphabetic operator is extremely general. Actually any processes of <u>information conversion</u> reduce to it or can be in some sense reduced to it. Here and in the future, by information we shall understand not only intelligent communications but in general any information on processes and states of any nature which can be detected by the sense organs of man or by instruments.

For certain specialized forms of information, for example information which is lexical or numerical, the alphabetic method of specification is the most natural and is constantly used. The transformations of these forms of information are reduced to the alphabetic operators in the most indirect fashion: both the input and the output information in any information converter in this case can be represented in the form of words, and the conversion of the information reduces to the establishment of some correspondence between the words. We recall that with rational expansion of the alphabet with words, account can be taken in the lexical information not only of ordinary words, but also entire sentences and even any sequences of sentences.

One of the characteristic tasks of the conversion of lexical information is the translation of texts from one language to another. It is well known that the translation problem does not reduce to the problem of establishing the correspondence between the words of the languages which are involved in the translation. If, however, we consider as words the entire books or at least individual sections of the book, then the problem of translation completely reduces to the problem of establishing correspondence between such generalized words. Thus, the problem of translation from one language to another can be treated as the process of the realization of some alphabetic operator.

It is worthy of note, moreover, that quite high-quality and grammatical translation permits, as is known, the possibility of known modifications of the translated text. Therefore the process of translation is described, not by the usual <u>single-valued</u> alphabetic operator, but by a multi-valued, or so-called <u>probabilistic</u>, alphabetic operator. Such an operator associates with each input word from the region of its definition not a single output word, but a whole ensemble of output words. In the specific application of this operator to a particular input word $\underline{p}$ there is a <u>random selection</u> of the output word from the ensemble of output words corresponding to the word $\underline{p}$.

In addition to the alphabetic operators for the translation from one language to another, we can construct alphabetic operators which resolve other problems of the conversion of lexical information, for example the problem of editing texts in a particular language, the problem of composing abstracts of articles, etc. It is not difficult to expand the field of application of the alphabetic operators, using the alphabetic representation not only for lexical information but also for other forms of information. For example, using the known techniques of chess notation, we can write chess positions in the form of words consisting of the letters of the Russian and Latin alphabets, digits, and punctuation marks (comma). In this case the process of the chess game can be interpreted as the process of establishing the correspondence between any given position and the position resulting from it after performing the next move. Thus, again in this case we are dealing with an alphabetic operator (probabilistic, generally speaking).

Similarly, it is not difficult to represent in the form of processes or realization of the alphabetic operators many other processes of information conversion, for example the orchestration of melodies, the solution of mathematical problems, the problem of production planning, etc.

It may seem at first that for the characterization of the conversion of continuous information (for example, visual or random auditory sensations) the concept of the alphabetic operator is insufficient. However this is not so, or more precisely, not entirely so.

The reception and conversion of continuous information is always accomplished with the aid of nonideal instruments which do not react to extremely small variations of the characteristics of the information being converted. In real instruments, detecting and converting continuous information, there always exist several limitations which make it possible to consider this information as alphabetic information. For greater clarity, let us consider visual information (the same phenomena occur with the other forms of specifying continuous information).

The first limitation is that of the resolving power of the instrument which receives the information. This limitation leads to the situation where sufficiently closely spaced points of the portion of space on which the information in question is distributed (for example, a picture or drawing) is sensed by the instrument (say, the human eye) as a single point. This implies the possibility of considering this information as information given, not at an infinite number of points, but only at a finite number of points.

The second limitation is associated with the limited sensitivity of the instrument receiving the information. This limitation leads to the instrument being able to distinguish only a finite number of levels of the quantity carrying the information (for example, the brightness of individual points of a drawing).

On the basis of the described limitations we come to the conclusion that the instrument, as a result of its nonideal nature, can at each given instant sense only one pattern of a finite (and not infinite as it might seem without account for the limitations indicated) number of different patterns of the instantaneous spatial distribution of the information in question.

Introducing for each such pattern a special literal notation, we come to the finite alphabet A which with account for the indicated limitations is completely adequate for the characterization of the information arriving at the input of the instrument (nonideal) which we are considering at every given instant of time. If we denote by the letter $\underline{n}$ the number of spatial points sensed by the instrument as individual points, and by the letter $\underline{m}$ the number of levels of the physical quantity carrying the information which are distinguished by the instrument, then the number of letters in the alphabet A will be equal, it is easy to see, to m^n (for simplicity we assume the number of levels which are distinguishable by the instrument to be identical for all points of the space).

Of course, the number of letters in the alphabet A which we have just estimated may be found to be excessively large (in the case of the reception of visual information by the human eye it may be estimated as a one with several thousand zeros following it). Nevertheless it is still finite, and from the abstract theoretical point of view the essential thing is only whether the alphabet A is finite or infinite.

Continuing our investigation, we note that every real instrument which receives and converts information has, along with the two limitations indicated, a third limitation. Here we are dealing with the limited passband of the instrument, which does not permit it to differentiate excessively rapid changes of the received quantities. In view of the familiar Kotel'nikov principle (Ref 46), the limitation of the pass band is equivalent to the introduction during the information transmission in place of the usual continuous time a conditional discrete time, neighboring instants of which differ from one another by quite definite (although usually very small) segments of time. Roughly speaking, as such an elementary segment of time we select the maximal segment in the course of which the instrument in question is incapable of differentiating the variations of the quantity carrying the information.

After the introduction of this descrete time, the information received by our instrument after any finite segment of time t naturally is represented in the form of a word in the previously introduced alphabet A. The number of letters in this word is equal to the number of instants $\tau_1,\ldots,\tau_k$ of the discrete time located in the given time segment t, and its i-th letter $(i = 1,2,\ldots,k)$ is the information received by the instrument at the instant of time τ_1 expressed in the form of a letter of the alphabet A.

Since analogous considerations are applicable not only to the input information but also to the output information, any real information converter must be considered (with account for the limitations indicated above) as an instrument realizing some alphabetic operator. The alphabetic operator realized by the instrument completely (with an accuracy to the information coding) determines the <u>informational essence</u> of this instrument, in other words the information conversion performed by this instrument.

Thus, we have established the extremely great generality of the concept of the alphabetic operator. Actually the theory of any information converter was found to reduce to the study of the alphabetic operators. And man encounters information converters literally at every step of his practical existence. The various instruments and devices for automatic control are information converters. Finally, one of the most important and essential aspects of the study of the activity of man himself is the aspect associated with consideration of man as a very complex and highly-perfected information converter. All this makes it possible to consider the theory of the alphabetic operators one of the most important component parts of cybernetics.

The basis of the theory of the alphabetic operators are the methods of representing them. In the case when the region of definition of definition of the alphabetic operator is finite the question of its representation, at least in the theoretical sense, is resolved very simply: the operator can be represented by a simple <u>correspondence table</u>. In the left side of such a table we write out all the words appearing in the region of definition of the operator in question, and in the right side we write the output words obtained as the result of the application of the operator to each word from the left side of the table.

Of course, if the region of the definition of the alphabetic operator is sufficiently large, this method of representation can become excessively cumbersome and therefore not applicable in practice. However, for the moment we shall not take such considerations into acount, limiting ourselves to the establishment only of the theoretical possibility of representing particular alphabetic operators.

In the case of an infinite region of definition of the alphabetic operator, its representation with the aid of a simple correspondence table becomes impossible in principle, since man does not have at his disposal the means to permit him to actually write out or perceive an infinite set of words. However, it is well known that man long ago learned to represent operators on infinite sets of words without writing out the entire correspondence tables. For this purpose it is sufficient to consider, for example, the alphabetic operator represented by the formula

$$\underbrace{xx\ldots x}_{n \text{ times}} \rightarrow \underbrace{yy\ldots y}_{n+1 \text{ times}} \quad (n=1,2,\ldots). \tag{4}$$

This formula defines the correspondence on an infinite set of words, achieved without actually writing out the entire correspondence table (which, of course, in this case cannot be done). In place of the correspondence table itself, this formula gives a rule with the aid of which, after a finite number of steps, there can be established the output word corresponding to any prescribed input word from the regeion of definition of the alphabetic operator being considered.

An analogous situation arises every time we need to represent an alphabetic operator with an infinite region of definition; in place of the correspondence table itself there is given a finite number of rules permitting after a finite number of steps the finding of the prescribed line of this table (the value of the alphabetic operator on any input word appearing in the region of its definition).

Alphabetic operators represented with the aid of finite systems of rules are customarily termed <u>algorithms</u>.

On the basis of the discussion above, we can easily understand that every alphabetic operator which can actually be represented is of necessity an algorithm. In particular, all alphabetic operators with finite regions of definition represented by (finite) correspondence tables will be algorithms. Formula (4) also represents an algorithm.

It is not difficult to construct other examples of algorithms. Associating with each whole positive number its square, we obtain an alphabetic operator in the alphabet consisting of all the digits of the number system used for the representation of these numbers. Since the rules for squaring make it possible after a finite number of steps to obtain the square of any prescribed whole number, this operator can be considered as an algorithm.

All the specific alphabetic operators considered in the present chapter (including the operators for translation from one language to another, chess moves, etc.) also can be represented with the aid of finite systems of rules and can, consequently, be considered as algorithms.

We must emphasize one distinction existing between the concepts of the alphabetic operator and the algorithm. In the concept of the alphabetic operator only the correspondence itself, established by the operator between the input and output words, is of essence, and not the method by which this correspondence is established. In the concept of the algorithm, on the other hand, the primary emphasis is placed on the <u>method of representation</u> of the correspondence established by the algorithm. Thus, the algorithm is nothing other than an alphabetic operator together with the rules defining its operation.

The concept of equality for the alphabetic operators and algorithms is defined in accordance with the foregoing. Two alphabetic operators are considered equivalent if they have the same region of definition and associate with any prescribed input word from this region identical output words. The concept of equality for algorithms includes the conditions of equality for the corresponding operators, but also provides for coincidence of the systems of rules which represent the operation of these algorithms on the input words. The algorithms for which there coincide only the alphabetic transformations (operators) defined by them, but, generally speaking, not the methods of representation, we shall term <u>equivalent algorithms</u>.

Usually in the abstract theory of algorithms we consider only those algorithms to which there correspond single-valued alphabetic operators. Every algorithm A of this kind differs in that to any input word <u>p</u> from the domain of its definition it associates a completely defined output word $q = A(p)$ regardless of the conditions in which the algorithm A operates. Such algorithms and their corresponding alphabetic operators will be called <u>determinate</u>.

In many cases it is advisable to expand the concept of the algorithm, introducing into the system of rules which describe the algorithms the possibility of the random selection of particular words or particular rules. Here the probability of a particular selection must be either fixed in advance or determined in the process of realization of the algorithm. Such algorithms will be called <u>random</u> and will lead to the multi-valued alphabetic operators. More precisely, for any input word <u>p</u> appearing in the domain of definition of the random algorithm A, this algorithm uniquely defines the probability $\alpha_p(q)$ of the appearance of the different output words q as the response to the input word p. The probabilities $\alpha_p(q)$ in the case of the usual random algorithm must not vary in the process of its functioning, although the algorithm itself can, of course, give different responses with repeated application to the same input word p.

We need to consider also the so-called self-variable algorithms, i.e., those algorithms which not only transform the input words applied to them but also themselves change in the process of this transformation. The result of the action of the self-variable algorithm A on a particular input word p depends not only on this word but also on the history of the preceding operation of the algorithm, i.e., on the (finite) sequence of input words processed by the algorithm A prior to the arrival at its input of the word p in question.

The generalization of the concept of the algorithm by means of the introduction of the possibility of self-variation is applicable to both the determinate and the random automata. In the latter case, depending on the history of the previous operation of the algorithm, there are changes of the probabilities $\alpha_p(q)$ of the different output words q associated by the algorithm A to any given input word p. This dependence can, moreover, also be expressed by a random function rather than a determinate one.

The self-variable algorithms are conveniently represented in the form of a system of two algorithms, the first of which, the so-called operational algorithm, performs the processing of the input words, and the second, termed the monitoring or controlling algorithm, introduces specific changes into the first, operational, algorithm. In Chapter 4 it is shown that the property of self-variability of the algorithm is determined not so much by the structure of the device which realizes the corresponding algorithm, as by the method of fractionation of the input information into individual words, which, as noted above, in the case of the abstract alphabets is to a considerable degree arbitrary. Thus, depending on the choice of this method the same device may in some cases realize a self-variable algorithm, in other cases it will realize a non-self-variable algorithm.

Throughout the first three chapters we shall consider only the conventional (determinate, non-self-variable) algorithms without making this stipulation in every instance. In the later chapters use will be made also of the generalized concepts of the algorithms introduced above.

§2. NORMAL ALGORITHMS

In this and the several following sections we shall study certain general methods of representation of the algorithms which are characterized by the property of <u>universality</u>, i.e., those methods which make it possible to obtain an algorithm which is equivalent to any prescribed algorithm. In this chapter rarious universal methods or representing algorithms are discussed, not in the historical sequency in which they were developed, but in an order which is most convenient from the point of view of the present volume. We begin our exposition with the so-called <u>normal algorithms</u> suggested and studied by Markov (Ref 53).

Every general method of representation of algorithms is termed an algorithmic system. The <u>algorithmic system</u> usually includes objects of a dual nature which, following Kaluzhnin (Ref 37), we shall term <u>operators</u> (or, more precisely, <u>elementary operators</u>) and <u>identifiers</u> (more percisely, <u>elementary identifiers</u>). Elementary operators are quite simple (simply represented) alphabetic operators whose sequential performance realizes any algorithms in the algorithmic system in question. The identifiers serve for the recognition of particular properties of the information processed by the algorithm and for the variation, depending on the results of the identification, of the sequence in which the elementary operations follow one another.

For indicating the set of elementary operators and the order of their sequencing one after the other in the representation of any specific algorithm, it is convenient to make use of the directed graphs of a special kind which, following Kaluhnnin (Ref 37), we shall term the graph-diagrams of the corresponding algorithms.

The graph-diagram of an algorithm is a finite set of circles (or other geometrical figures), termed the elements of the graph-diagram, which are interconnected by arrows. To each element, other than the two special elements which are termed the input and output, there is associated some elementary operator or identifier. From each element representing an operator, and also from the input element, there emerges precisely one arrow; from each element representing an identifier there emerge precisely two arrows; no arrow emerges from the output element. Any number of arrows can enter an element.

The algorithm defined by any given graph-diagram operates as follows. The input word enters first the input element and travels in the directions indicated by the arrows, being transformed on passage through the operator elements by the operators associated with these elements. When the word enters an identifying element a check is made of the condition associated with this element (application of conditional identifier). If the condition is satisfied, the word emerges from the element along one of the arrows (usually indicated by the symbol "+"), and if the condition is not satisfied it emerges along the other arrow (indicated by the symbol "-").

The word is not altered in the identifying elements. If the input word p applied to the input element of the graph-diagram, after passing through the elements of the diagram and being transformed, arrives after a finite number of steps at the output element, it is considered that the algorithm is <u>applicable</u> to the word *p* (the word *p* is in the <u>domain of definition</u> of this algorithm), and the result of the action of the algorithm on the word *p* will be that word which is in the output element of the diagram. If after the application of the word *p* to the input element of the graph-diagram its transformation and movement along the graph-diagram lasts infinitely long, without arrival at the output element, then it is considered that the algorithm is <u>not applicable</u> to the word *p*, in other words, the word *p* is not in the domain of definition of the algorithm.

In normal algorithms use is made only of one type of elementary operator, termed <u>substitution operators</u>, and one type of elementary identifier, termed <u>occurrence identifier</u>. We shall describe these identifiers and operators in more detail. To do this we shall first acquaint ourselves with the concept of <u>occurrence</u> of one word in another.

Let *p* and *q* be two arbitrary words in a particular alphabet. We say that the word *q* occurs in the word *p* if the word *p* can be represented in the form $p = p_1qp_2$, where p_1 and p_2 and some words, possibly even empty ones. The occurrence found for the word *q* in the word *p* is termed <u>first left</u> (or simply <u>first</u>) <u>occurrence</u> if in the considered representation of the word *p* in the form $p = p_1qp_2$ the word p_1 has the shortest possible length among all similar representations of the word *p*.

The occurrence identifier is given by the indication of some fixed word *q*, and the sense of its application is that for any given word *p* a check is made of the condition of whether or not the word *q* occurs in the word *p*. The substitution operator is usually given in the form of two words connected by an arrow, $q_1 \rightarrow q_2$. The operation of the operator amounts to performance of the substitution of the word q_2 in place of the first left occurrence of the word q_1 in any given word $\underline{p}$. If we separate explicitly the first occurrence of the word q_1 in the word $\underline{p}$, writing the word $\underline{p}$ in the form $p_1q_1p_2$, after the application of the considered operator it is transformed into the word $p_1q_2p_2$.

In the application of the occurrence identifier we agree to separate the found (first left) occurrence of the identified word in the given word by the use of parentheses. For example, applying to the word p = xxyxyxx the occurrence identifier of the word q = xy, we separate the first occurrence of the word $\underline{q}$ in the word $\underline{p}$ as follows: p = x(xy)xyxx.

The algorithms which are represented by graph-diagrams consisting exclusively of word occurrence identifiers and substitution operators are termed <u>generalized normal algorithms</u>. Here it is assumed that to each substutution operator of the form $q_1 \rightarrow q_2$ there is connected only a single arrow: an arrow with a "+" sign emerging from the q_1 identifier.

An example of a graph-diagram of a generalized normal algorithm is shown in Fig.1. On this figure the identifiers are shown in the form of rectangles. The operator xy → denotes substitution of an empty word in place of the first occurrence of the word xy. In accordance with the notation of the empty word which was used in the preceding section, this operator can be written also in the form xy → e.

Considering the operation of the algorithm A given by the graph-diagram of Fig. 1, we note that the first operator from the top performs the transposition of $\underline{x}$ to the left and of $\underline{y}$ to the right portion of the word until the word takes the form xx...xyy...y (all $\underline{x}$ precede all $\underline{y}$). Only after reduction of the word to this form does the second operator come into action, annihilating the pairs xy until only $\underline{x}$ or $\underline{y}$

remain in the word. If in the originally given word p there were m x's and n y's, then as a result of the operation of the algorithm A it is transformed into the word $q = A(p)$, having the length $|m-n|$ and consisting of only x's (if $m > n$) or only y's (if $n > m$).

Having considered the generalized normal algorithms, let us turn to the characteristic of the normal algorithms themselves. Those generalized normal algorithms whose graph-diagrams have some special form are termed normal algorithms. In order to describe this form we note that as a result of the definition of the generalized normal algorithms presented above, every operator $q_1 \rightarrow q_2$ occurs paired with the identifier q_1 in the graph-diagram of such algorithms.

Let us combine in the graph-diagram each such pair of elements into a single element, retaining for it the notation of the corresponding operator.

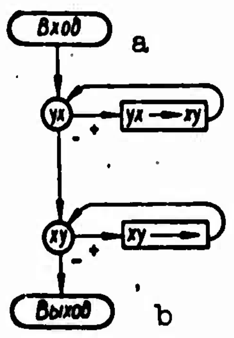

Fig. 1. a) Input;
b) output.

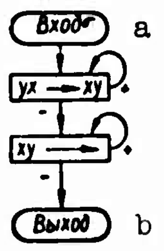

Fig. 2. a) Input;
b) output.

From each combined element there will emerge two arrows: an arrow with the symbol "+" along which there is directed the word subjected to the action of the operator of the given element, and an arrow with the symbol "–" along which the word is directed if the element operator is not applied to it. Nonapplicability of the substitution operator to a word denotes the absence of the occurrence of the left portion of the operator (the word q_1 in the operator $q_1 \rightarrow q_2$) in the given word.

Using the described technique for combining elements, the graph-diagram of the algorithm shown in Fig. 1 can be represented in the diagram shown in Fig. 2. Such a graph-diagram with combined elements in the case of the normal algorithms must satisfy the following conditions:

a) all the combined (operator-identifier) elements of the graph-diagram are ordered by means of assigning them the sequential numbers from 1 to n, and a negative output (arrow with symbol "−") of the i-th element is connected to the (i + 1)-th element ($i = 1, 2, \ldots, n - 1$) and a negative output from the n-th element is connected to the output element of the graph-diagram;

b) the positive outputs (arrows with the symbol "+") of all the combined elements are connected either to the first or to the output element of the graph-diagram. In the first case the substitution of the operator of the corresponding element is termed <u>ordinary</u>, in the second case it is termed <u>final</u>.

c) the input element is connected by an arrow to the first combined (identifier-operator) element.

These conditions are necessary and sufficient for the graph-diagram which satisfied them to represent an ordinary normal algorithm rather than a generalized normal algorithm. It is easy to verify that the graph-diagram shown in Fig. 2 is not a graph-diagram of a normal algorithm since it does not satisfy the second of the conditions just formulated (condition "b").

The normal algorithms are customarily represented not by graph-diagrams but simple by the ordered set of substitutions of all the operators of the given algorithm, termed the diagram of the given algorithm. Here the ordinary substitutions are written, as shown above, in the form of two words connected by an arrow ($q_1 \rightarrow q_2$) while the final substitutions are designated by an arrow with a dot ($q_1 \rightarrow .q_2$).

The order of performance of the substitutions is completely determined after this by the conditions "a", "b" and "c". Actually, as a result of these conditions the arbitrary i-th substitution of the algorithm diagram must be performed in, and only in, the case when it it the first of the applied substitutions (all substitutions from the 1-st to the (i - 1)-th not applied). The process of performing the substitutions is terminated only when none of the substitutions of the diagram is applicable to the word obtained or when some final substitution is performed (for the first time).

As an example, let us consider the operation of the normal algorithm A given by the diagram

$$\begin{gathered} yyx \rightarrow y, \\ xx \rightarrow y, \\ yyy \rightarrow \cdot x. \end{gathered}$$

Let us assume that we are given the input word p = xyxxxyy. The first substitution of the algorithm A is not applicable to this word, in order to apply the second substitution we isolate the first occurrence of its left part (xx) in the word p:p = xy(xx)xyy. After performance of the second substitution of the algorithm, we obtain the word p_1 = xyyxyy, to which the first substitution of the algorithm is applicable: p_1 = x(yyx)yy $\rightarrow$ xyyy = p_2. Only the third substitution is applicable to the resulting word: p_2 = x(yyy) $\rightarrow$ xx = p_3, and since it is denoted as a final substitution, the word p_3 is the final result of the action of the algorithm A on the original word p, i.e., p_3 = A(p).

If the third substitution of the algorithm A were not a final substitution, then the process of substitution could be continued and in place of the word p_3 = xx we would obtain the word p_4 = y as the result of the action of the algorithm on the original word p.

The use of final substitutions in the normal algorithm diagrams along with the ordinary substitutions is necessary in order to have the possibility of realizing in such diagrams the arbitrary <u>constructive</u> alphabetic operators, i.e., those alphabetic operators which are determined with the use of a finite number of rules. Actually, any normal algorithm A whose diagram does not contain a single final operator can terminate its operation only when none of its substitutions is further applicable. This implies directly that repeated application of algorithm A to the word A(p) obtained as a result of the application to any input word p cannot change this word. In other words, the following identity relation (valid for any input word p) is satisfied for the algorithm A (see Markov [53]):

$$A(A(p)) = A(p). \qquad (5)$$

By no means every constructive alphabetic operator satisfies this relation. An example of an alphabetic operator for which relation (5) is not valid is the operator B, whose action on any word p amounts to prefixing some fixed letter x to the left of this word: B(p) = xp. From what we have said above it is clear that this operator cannot be realized by the use of a normal algorithm whose diagram does not contain final substitutions.

At the same time it is easy to verify that this operator is realized by the normal diagram consisting of the single final substitution $\rightarrow \cdot x$ (or, what is the same, $e \rightarrow \cdot x$). Actually, as a result of the definition of occurrence taken above, an empty word occurs in every word p, and its first occurrence will not have a single letter on its left. It follows directly from this that the use of this substitution on the arbitrary word p converts it to the word xp.

It is no less evident that in the construction of the theory of normal algorithms we cannot limit ourselves to only final substitutions.

Actually, the normal algorithm whose diagram consists only of final substitutions operates on each input word p with no more than one of these substitutions, after which the required output word A(p) is obtained immediately. In view of the finiteness of the algorithm diagram, the moduli of the differences of the lengths of the words p and A(p) are bounded in the aggregate (for any selection of the input word p) by the same number N (the maximum of the moduli of the differences of the lengths of the words in the left and right sides of the substitutions of algorithm A).

There do exist, however, simple constructive algorithms for which the moduli of the differences of the lengths of the input and corresponding output words are not bounded in the aggregate. An example of such operators might be the operator D for the doubling of the input words, whose action on any input word p is determined by the equality D(p) = pp. From what we have said above, it is clear that the representation of this operator in the form of a normal algorithm whose diagram contains only final substitutions is obviously impossible.

Thus, if we present to an algorithmic system based on the use of normal algorithms the requirement of universality (possibility of constructing a normal algorithm which is equivalent to any a priori specified algorithm), then a necessary condition for such universality is the use of both forms of substitutions, both final and ordinary. This condition is also sufficient, i.e., we can formulate the normalization principle (see Ref. 53).

Normalization principle. For any algorithm (constructively given alphabetic representation) in the arbitrary finite alphabet A we can construct an equivalent normal algorithm on the alphabet A.

The concept of a normal algorithm on an alphabet which is used on the formulation of the normalization principle means the following. In many cases it is not possible to construct a normal algorithm equivalent to a given algorithm (in the alphabet A) if we use only letters of the alphabet A in the substitutions of the algorithm. However, we can construct the required normal algorithm by adding to the alphabet A some number of new letters or, as we usually say, performing an _expansion_ of the alphabet A. In this case it is customary to say that the constructed (normal) algorithm is an _algorithm on_ the alphabet A. We agree, however, that in spite of the expansion of the alphabet the algorithm will as before be applied only to words in the original alphabet A.

As shown by Markov [53] and Nagornyy [58], if we can construct the normal algorithm equivalent to a given algorithm in the alphabet A by joining to the alphabet A some (possibly very large) finite number of letters, then we can construct its equivalent normal algorithm by adjoining to the alphabet A only a single additional letter.

It is not possible to give a rigorous mathematical proof of the normalization principle, since the concept of the arbitrary algorithm is not a rigorously defined mathematical concept. Therefore, we must approach its substantiation just as we approach the substantiation of every law or principle of natural science. The substantiation which we can give the normalization principle in this framework makes it possible to consider this principle credible to a very high degree. We shall indicate the basic processes of this substantiation. In order to simplify the formulations, we shall agree, following Markov [53], to term a particular algorithm _normalizable_ if we can construct its equivalent normal algorithm (using, possibly, expansion of the alphabet) and term it unnormalizable otherwise. We can now state the normalization principle in a somewhat altered form.

All algorithms are normalizable.

The validity of this principle is based first of all on the fact that all the algorithms known at the present time are normalizable. Since in the course of the long history of the development of the exact sciences a considerable number of different algorithms have been devised, this statement is convincing in itself.

In actuality it is even more convincing. We can show that all the methods known at the present time for the composition of algorithms which make it possible to construct new algorithms from the already known ones do not go beyond the limits of the class of normalizable algorithms. In other words, if the original algorithms were normalizable, then any compositions of these algorithms (among the number of forms of compositions known at the present time) will also be normalizable. This implies that for the construction of an example of an unnormalizable algorithms it is necessary to use techniques which are qualitatively different from everything the mathematician has encountered up till now.

However this is not all. A whole series of scientists have undertaken special attempts to construct algorithms of a more general form and all these attempts have not been carried beyond the limits of the class of normalizable algorithms. We shall consider one of these attempts (the algorithmic scheme of Kolmogorov-Uspenskiy) below. The failure of these attempts is in itself the most striking evidence in favor of the validity of the normalization principle.

Thus the normalization principle should be considered sufficiently substantiated, although this substantiation does not exclude completely the possibility of its refutation in the future (by construction of an example of an unnormalizable algorithm). In any case, the normalizable algorithms encompass a significant portion of the algorithms (if not all) and therefore the system of normal algorithms can be considered in practice to be a universal algorithmic system.

Let us consider now some of the common forms of compositions of algorithms which were mentioned above. We shall define not the composition of the algorithms themselves, but the composition of their corresponding alphabetical representations, however, as remarked above, the possibility of normalization of the result of the composition of the normal algorithms makes it possible (at least in the class of normal algorithms) to extend the definition of the composition of the representations to the composition of the algorithms themselves.

One of the most common forms of composition of algorithms (representations) is the superposition of algorithms. In the superposition of the two algorithms A and B the output word of the first algorithm (A) is considered as the input word of the second algorithm (B), so that the result of the superposition of the algorithms A and B can be represented in the form D(p) = B(A(p)). This definition extends to the superposition of any finite number of algorithms.

A superposition of generalized normal algorithms can be considered an a generalized normal algorithms. For this it is sufficient that the output element of the graph-diagram of each preceding algorithm be combined with the input element of the succeeding algorithm. The normalization of a superposition of normal algorithms requires considerable skill, however it too can always be accomplished [53].

We shall point out some other forms of compositions of algorithms.

The union of the algorithms A and B in the same alphabet $\dot{X}$ is the term given to the algorithm C in the same alphabet which transforms any input word $\underline{p}$ contained in the intersection of the domains of definition of the algorithms A and B into the words A(p) and B(p) written side by side; this algorithm is considered undefined on all the remaining input words.

A ramification of algorithms is a composition of the three algorithms A, B and C. Designating the result of this composition by D, we shall consider that the domain of definition of the algorithm D coincides with the intersection of the domains of definition of all three algorithms A, B and C, and that for any word p from this intersection $D(p) = A(p)$ if $C(p) = e$, and $D(p) = B(p)$ if $C(p) \neq e$.

A repetition (iteration) is the composition of the two algorithms A and B. Designating the result of this composition by P, we define that for any input word q the corresponding output word P(q) is determined by the following condition: there exists such a series of words $q = q_0, q_1, q_2, \ldots, q_n = P(q)$, that for all $i = 1, 2, \ldots, n q_i = A(q_{i-1})$, for all $i = 1, 2, \ldots, n - 1$ $B(q_i) \neq e$, and $B(q_n) = e$. In other words, the algorithm A is applied sequentially several times until a word is obtained which is transformed by the algorithm B into the empty word e (we can, of course, select any other fixed ford rather than the empty word).

All the methods described for the composition of the normal algorithms lead to normalizable algorithms [53].

Of very great importance for the normal algorithms, just as for every universal algorithmic system, is the problem of the construction of the so-called universal algorithm. Let us consider the universal algorithm in application to the normal algorithms.

Let us be required to construct a normal algorithm which will perform the operation of any normal algorithm if we are given the diagram (substitution set) of this latter algorithm.

The exact formulation of the problem on the universal algorithm can be accomplished by various methods. We shall describe one of the most natural methods for such a formulation. To do this we first of all fix some standard alphabet $\mathfrak{X}$ (for example, binary). For all other possible alphabets we fix some definite method of coding the letters of these alphabets in the selected standard alphabet. In the case of the binary standard alphabet this can be done, for example, as follows: the letters of any given alphabet are numbered sequentially using the natural numbers, after which the i-th letter is assigned the binary code, beginning and ending with zero and having between these zeros exactly i ones. If the total number of letters in the given alphabet is equal to n, then we introduce also the additional ((n + 1)-st, (n + 2)-nd, etc.) letters for the designations of the symbols used in the diagrams of the normal algorithms (arrows, dots, separation sign between formulas) and also for the designation of the special end sign which stands at the beginning and end of the algorithm diagram.

After writing the algorithm diagram with a single word and coding the letters of this word by the method just described, we obtain a word in the standard alphabet, which is termed the transform of the given algorithm. For example, for the normal algorithm given by the diagram

$$\begin{matrix} xy \rightarrow x \\ y \rightarrow \cdot, \end{matrix}$$

the transform A^u of the algorithm A in the binary alphabet can be obtained as follows: we fix the numberation of the letters, considering x to be the first, y the second, the arrow to be the third, the dot to be the fourth, the separation symbol to be the fifth, and the end symbol to be the sixth letter. Then the transform A^u of the algorithm A is written as: 060 010 020 030 010 050 020 030 040 060. Here, for brevity, in place of writing out in a row any positive number n of ones we have written this number n itself.

Along with the transform of the algorithm A, there can also be obtained by use of the coding in the standard alphabet $\ddot{\mathfrak{X}}$ described above the transform p^u of any input word $\underline{p}$ of this algorithm.

The following theorem on the universal normal algorithm is valid (see Markov [53]).

There exists such a normal algorithm U, termed a universal normal algorithm, which for any normal algorithm A and any input word $\underline{p}$ from the domain of definition of this latter algorithm transforms the word $A^u p^u$, obtained by suffixing the transform of the word $\underline{p}$ to the transform of the algorithm A, into the word which is the transform of the corresponding output word A(p) into which the algorithm A transforms the word $\underline{p}$. If, however, the word $\underline{p}$ is chosen so that the algorithm A is not applicable to it, then the universal algorithm U is not applicable to the word $A^u p^u$.

This theorem is of tremendous value, since it implies the possibility of the construction of a machine which can perform the operation of any normal algorithm, which means, in view of the normalization principle, the operation of any arbitrary algorithm. For this purpose it is sufficient to insert into the machine a program, i.e., the transform of that normal (normalized) algorithm whose operation the machine is to perform.

However, although in principle the possibility has been proved of the normalization for all the algorithms known at the present time, the actual performance of the normalization is a very serious matter even for the relatively simple algorithms (the algorithm for the multiplication of two whole numbers, for example). This means that the programming for a machine simulating the universal normal algorithm would be excessively unwieldy and impractical. Therefore, in practice the machines which make possible the realization of the operation of any algorithm are designed on the basis of the use of other algorithmic systems which differ from the system of the normal algorithms. These systems are described in Chapter 5.

§3. THE KOLMOGOROV-USPENSKIY ALGORITHMIC DIAGRAM

The present section describes the method suggested by Kolmogorov and Uspenski [43] for the determination of algorithms of the most general form. For the construction of the corresponding algorithmic diagram they choose the method which is based only on those properties which are without question inherent to any algorithmic diagram and which will realize these properties in particular specific forms without permitting any loss of generality in doing so.

In the construction of such a generalized algorithmic diagram it is useful to picture as a visualizable model a man who is performing the computation or other processing of information in accordance with a particular precisely prescribed system of rules. The man performs the role of information converter, while the converted information itself is located outside of the man. We shall assume for definiteness that this information is written on sheets of paper, and that the man has at his disposal an unlimited supply of clean sheets and an unlimited reserve of space for storage of filled-out sheets. The transformation of the information realized by the man is broken down into individual discrete steps. At each such step the man surveys some number of completed sheets and, depending on the contents of these records, using a strictly defined and time-invariant system of rules located in his memory, he performs certain alterations in the reviewed information. These alterations may be of three forms: erasure (annihilation) of the entire reviewed information or some portion of it, recording on the reviewed sheets of new information, alteration of the ensemble of reviewed sheets.

At first glance it seems that the requirement for the invariance in the system of rules used for the performance of the processing of the information significantly narrows the range of problems considered in comparison with the problems which can in actuality be solved by man, since man is capable of altering the rules in the course of the operation. In actuality this limitation is not significant, since the nature of the alteration of the information at each step of the processing depends not only on the rules of the transformation but also on this information itself. In this connection it is possible in case of necessity to vary the nature of the information transformation with the course of time, to introduce corresponding changes in the information itself, and not in the rules stored in the memory of the processor, in other words, to write down in the rules on the sheets of paper the required alterations and not to memorize them.

An absolutely necessary limitation in the design of any algorithmic system is the capability of the information processor to absorb at any given instant of time only a limited quantity of information. If the total volume of the material being processed exceeds the volume of this active zone of the processor, then the information must be brought into the processing gradually, step by step.

After these preliminary remarks we turn directly to the description of the Kolmogorov-Uspenskiy diagram. The information in this diagram, as in general in the case of the alphabetic conversions, is written with the aid of a finite number of symbols, letters, which we shall designate as T_0, T_1, ..., T_n. To achieve the greatest possible generality, we shall also establish certain relations between the symbols, these relations belonging to one of the types R_1, R_2, ..., R_m. For each type of relation R_i we fix the number k_i of related symbols (letters). We designate by K the maximal number among the numbers k_1, ..., k_m. The relations between the symbols are introduced in order to take account of the case of complex letters which designate, for example, entire pharases in ordinary language. In that case the composition of the phrase (letter) may include indications of the relative positioning of information (other letters) which has direct relation with the letter in question (say, information which must be brought into consideration in the following step of the algorithm). The limiting of the number of related symbols depends on the boundedness of the information contained in each letter (otherwise the letter cannot be contained entirely in the active zone and it must be divided into individual portions).

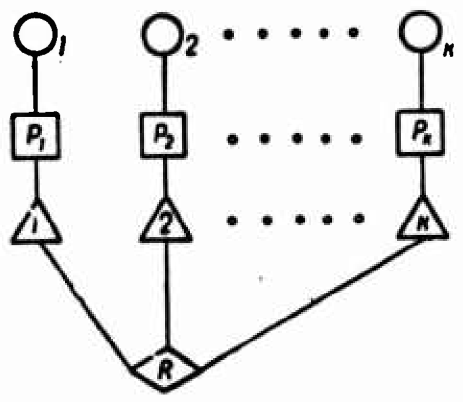

Fig. 3.

Let us assume that all the relations in which any given letter can occur are ordered in some way and numbered, and that the total number of such relations is bounded by the same number <u>s</u>. We shall use circles to designate the letters, introducing when necessary numeration of these circles with numerals written adjacent to the corresponding circles. These numerals have no relations to the type of symbol (letter) designated by the given circle. When necessary, the symbol of the corresponding letter is written inside the circle which represents it.

Any relationship between sumbols (letters) can now be represented as shown in Fig. 3.

The subscripts of p_1, p_2, ..., p_k on this figure show the position occupied by the relation in question in the ordered set of relations for the corresponding (designated by the numbered circles) letter. These subscripts (regardless of the choice of the letters and the form of the relation R) can take only the values 1, 2, 3, ..., s.

We can considerably simplify the writing of the information in this diagram by adding to the number of letters T_0, T_1, ..., T_n s + K + m letters: s letters for the designation of the numbers of relations in any given element (squares in Fig. 3), K letters for the designation of the numbers of relations with the letters for any given relation R (triangles in Fig. 3), and m letters for the designation of the R_1, R_2, ..., R_m relations themselves. If we denote all the new letters by circles, then the information takes the form of a set of circles connected between one another by paired bonds. Then there is no requirement for any special numeration for the order of occurrence of a letter in particular relations, since, as shown in Fig. 3, all the letters related with any single letter will inevitably be different. Thereby the relations in which a given symbol (letter) occurs are numbered automatically--by the numbers of the sumbols (letters) with which the given symbol is related.

Thus, finally, the information in the written algorithmic diagram is represented by an arbitrary finite set M whose elements are the fixed letters T_0, T_1, ..., T_N ($N \geq 1$) where in the set M each of the letters T_2, T_3, ..., T_N can occur any number of times, and, in addition, in the set there occurs each time one and only one of the letters T_0 or T_1. On this set there is established a paired relation (certain letters "join" pairwise with one another) so that the following condition "a" is satisfied: all the letters connected with any single letter of the set M are pairwise different.

In other words, the information is in the form of some one-dimensional complex (linear undirected graph), whose vertices (designated by the circles) are identified with the letters T_0, T_1, ..., T_N and the (undirected) lines connecting certain pairs of vertices are identified with the paired relations between the letters described above.

The requirement for the occurrence in the complex in question (the set M) of one and only one vertex identified either with the letter T_0 or with the letter T_1 is associated with the necessity for the establishment of the reference point (center of the active zone) of the information, and one of these letters (we assume that it is the letter T_0) is required for the compleses designating the information whose processing is not yet completed, and the other (in the present case the letter T_1) is required for the complexes designating the terminal information from which the final results of the operation of the algorithm must be extracted.

The vertex of the informational complex S, which is identified with the letter T_0 or T_1, is termed the initial vertex of the complex. The active zone of the complex S is the subcomplex of the complex S which consists of the vertices (letters) and the lines (relations) belonging to the chains of length $\lambda \leq P$ containing the initial vertex, where P is a number which is determinate, fixed for the given algorithm. Here and hereafter we use the term chain to designate any finite sequence of vertices $B_1, B_2, \ldots, B_p$ such that any two neighboring vertices in this sequence are connected by lines; the number of all these vertices (equal to $p-1$) is termed the length of the chain, and these lines themselves are also included in the chain in question.

The ensemble of all the vertices of the active zone of the information complex which are connected with the initial vertex by chains of length P and are not connected with it by chains of lesser length is termed the boundary of this zone. The complex is called bound if any two of its vertices can be connected by a chain. The ensemble of vertices and lines lying beyond the limits of the active zone of the complex S is termed the external portion of the complex.

Two complexes are termed mutually isomorphic if between their vertices we can establish mutually single-valued correspondence, where the corresponding vertices are designated by the identical letters T_0, T_1, ..., T_N, and corresponding pairs of vertices are either simultaneously connected or simultaneously not connected to one another. Mutually isomorphic complexes are in essence identical, and differ perhaps only in the method of their representation (position of the vertices on the plane, for example).

In view of the boundedness of the total number of vertices in the active zone of the information complex of any given algorithm and the boundedness of the number of letters T_0, T_1, ..., T_N for any given algorithm A, there exists only a finite number of different (pairwise nonisomorphic) active zones U_1, U_2, ..., U_r. Starting from this, the rules for their processing can be given by the simple correspondence table $U_i \rightarrow W_i$ $(i = 1, 2, \ldots, r)$.

The complexes appearing in the right side of this table must have subcomplexes which are isomorphic to the boundaries of the corresponding active zones U_i, and these isomorphisms must be fixed once and for all. In other words, to each vertex lying on the boundary $L(U_i)$ of the active zone U_i there must be associated a completely determined vertex of the complex W_i $(i = 1, 2, \ldots, r)$. Each of the complexes W_i must satisfy all the conditions imposed above on the information complexes; in particular, it must have one and only one initial vertex, designated by the letter T_0 or by the letter T_1.

With the aid of the constructed correspondence table, we determine the operator R_A which performs the direct processing of the information complex at each step of the operation of the given algorithm A. In the considered information cemplex S (initial and intermediate), we find the initial vertex. Drawing from it all possible chains of length

P, we construct the active zone and determine its boundary L(U).

Further, we find that (single) active zone from the left side of the correspondence table which is isomorphic to the found active zone U. As a result of the properties defined above of the information complexes (in particular the property "a") and the connectedness of the two complexes U_1 and U with one another, only one isomorphism is possible. This makes possible unique identification of the vertices lying on the boundary L(U) of the active zone U with the corresponding vertices, for the isomorphic case, lying on the boundary $L(U_1)$ of the active zone U_1 and, using the identification of the vertices employed in the correspondence table, also with certain vertices of the complex U_1.

Now it is easy to remove all the interior, i.e., not lying on the boundary L(U), portion of the active zone U and replace it by the subcomplex W'_1 of the complex W_1 which includes all the elements of this complex except its vertices which were identified earlier. Thus, we "insert" into the information complex in question the new complex W'_1 in place of the internal portion of its active zone while retaining unchanged the boundaries of the active zone.

Since in the complex W'_1 the initial vertex occupies a new position with relation to the boundary of the previous active zone, the new active zone, determined after the insertion, will have a different boundary. The new information complex S' obtained after such an insertion then will be the result of the application of the direct processing operator R_A of the algorithm A in question to the original information complex S. The direct processing operator is applied to the resulting information complex until obtaining a complex whose initial vertex is designated by the letter T_1 and not by the letter T_0.

Such a complex is termed a <u>terminal complex</u> and its maximal bound subcomplex, containing the initial vertex T_1, is considered to be the solution, i.e., the information complex obtained as the result of the action of the algorithm A on the initial (input) information complex S_0. If, however, the algorithm continues operation without end without obtaining a terminal complex at any step, then, just as in the case of the normal algorithms, we take it that the algorithm in question is not applicable to the given initial complex S_0.

We can expand the definition of the algorithm so as to permit in the right side of the table correspondences of a complex without an initial vertex. The application of the substitution with such a right side leads to natural termination of the algorithmic process, since the determination of the active zone and the further substitution become impossible.

However, since the terminal complex (in the sense defined above) does not appear, again in this case the algorithmic process must be considered to have terminated without result and the algorithm is considered inapplicable to the corresponding initial information complex.

Still another type of unsuccessful termination of the algorithmic process is possible in which the correspondence table does not contain all forms of active zones which are possible for the given algorithm. In the case when the information complex reaches a state in which although there is an initial vertex designated by the letter T_0 none of the substitutions of the correspondence table are applicable, it is also considered that the algorithm is not applicable to the corresponding initial information complex.

We must make still one more remark on the nature of the substitutions in the correspondence table. If special measures are not taken, as a result of the substitutions the condition "a" introduced above may be violated; this condition must be satisfied by all the information complexes we are considering. In order to avoid such a distortion of the information, it is clearly sufficient to assume that any vertex of the arbitrary complex W_1 from the right side of the correspondence table, which in the process of "insertion" is identified with some vertex $\underline{g}$ of the boundary of the active zone in the complex W_1, can be connected by lines only with the initial vertex and with the vertices designated by the same letters as the vertices with which there is connected by lines in the complex U_1 the vertex corresponding to the vertex $\underline{g}$.

This condition (we term it "b") does not violate the generality of our considerations. The boundary used in performing the "insertion" operation is defined quite arbitrarily. If we included in the boundary not only those vertices which are removed from the initial vertex by the distance P (connected with it by chains of length P but not by chains of lesser length) but also the vertices which are removed from it by the distance P — 1, then, establishing the isomorphism of the boundaries in the compleses U_1 and W_1 we would obtain, as it is not difficult to see, a stronger limitation on the correspondence table than the limitation imposed by the condition "b".

Careful analysis of the description of the Kolmogorov-Uspenskiy algorithmic diagram shows that in form this diagram to a very significant degree is reminiscent of the operation actually performed by a man when he processes information supplied to him externally in accordance with the particular rules of an algorithm which he has memorized. The developers of this diagram took special measures not to lose generality in the nature of the transformation performed. Nevertheless, they demonstrated that the diagram which they described gives the possibility of constructing only normalizable algorithms. This result can be considered confirmation of the normalization principle formulated in §2.

§4. OT' 'HEORETICAL ALGORITHMIC SYSTEMS

I...orically the first algorithmic system which received fairly complete and thorough development was the system based on the use of constructively determinate arithmetic (integral) functions which were given the name recursive functions. The use of these functions in the theory of algorithms is based on the idea of numeration of the words in any alphabet by means of the sequential natural numbers. This numeration can be accomplished most simply by arranging the words in increasing order of their lengths, and arranging words having the same length in an arbitrary (lexicographic, for example) order.

After numeration of the input and output words in an arbitrary alphabetic operator, this operator is transformed into the operator $y = f(x)$ in which both the argument x and the function y itself take nonnegative integral values. The function $f(x)$, of course, can not be defined for all values of the argument x but only for certain values of x which constitute the domain of definition of this function. Such partially defined integral and shole-valued functions are usually termed arithmetic functions for brevity.

Among the arithmetic functions we separate the following particularly simple functions which we shall term elementary arithmetic functions: the function identically equal to zero (defined for all whole nonnegative values of the arguments); the identity functions $f(x_1) = x_1$, which repeat the values of their arguments; the direct succession function $f(x) = x + 1$, which also defined for all whole nonnegative values of its argument.

Using as original functions the elementary arithmetic functions just listed, we can with the aid of a small number of general constructive techniques construct ever more and more complex arithmetic functions. In the theory of recursive (constructive arithmetic) functions three operations are of particularly great importance: superposition, primitive recursion and least root operations.

The operation of superposition of functions involves the substitution of some arithmetic functions in place of the arguments of other arithmetic functions. Thus, from the already known functions we can construct new arithmetic functions. For example, performing the superposition of the functions $f(x) = 0$ and $g(x) = x + 1$, we arrive at the function $h(x) = 1$. With the superposition of the function $g(x)$ with itself there appears the function $p(x) = x + 2$, etc.

The operation of primitive recursion makes it possible to construct an n-place arithmetic function (function of n arguments) from two given functions, one of which is $(n - 1)$-place, and the other is $(n + 1)$-place. the method of this construction is determined by the following two relations:

$$f(x_1, \ldots, x_{n-1}, 0) = g(x_1, \ldots, x_{n-1}); \qquad (6)$$

$$f(x_1, x_2, \ldots, x_{n-1}, x_n + 1) = h(x_1, x_2, \ldots, x_n, y), \qquad (7)$$

where $y = f(x_1, \ldots, x_{n-1}, x_n)$; f is the function being determined and g and h are the given functions.

For a proper understanding of the operation of primitive recursion we must note that every function of a smaller number of variables can be considered as a function of any larger number of variables. In particular, constant functions, which it is natural to consider as functions of a zero argument, can if desired by considered as functions of any finite number of arguments.

As an example, let us consider how the operation of primitive recursion is applied to construct from the elementary arithmetic functions the two-place summation function $f(x,y) = x + y$. This function is determined with the aid of the identity function $g(x) = x$ and the direct succession function $h(x) = x + 1$

$$f(x, 0) = x = g(x);$$
$$f(x, y+1) = (x+y)+1 = h(f(x, y)).$$

We can construct similarly the product, exponential, power and other widely known arithmetic functions.

The functions which can be constructed from the elementary arithmetic functions using the operations of superposition and primitive recursion any (finite) number of times in any sequence are termed the primitively recursive functions.

The majority of the arithmetic functions are primitively recursive functions. Nevertheless the primitively recursive functions do not include all the arithmetic functions which can be defined constructively. In the construction of all these functions use is made of other operations, in particular the least root operation.

The least root operation makes it possible to determine a new arithmetic function $f(x_1, \ldots, x_n)$ of n variables with the aid of the previously constructed arithmetic function $g(x_1, \ldots, x_n, y)$ of $n + 1$ variables. For any given set of values of the variables $x_1 = \alpha_1, \ldots, x_n = \alpha_n$ as the corresponding value $f(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of the function being determined $f(x_1, x_2, \ldots, x_n)$ we take the least integral nonnegative root $y = \beta$ of the equation $g(\alpha_1, \ldots, \alpha_n, y) = 0$. In the case of nonexistence of integral nonnegative roots of this equation, the function $f(x_1, x_2, \ldots, x_n)$ is considered indeterminate for the corresponding set of values of the variables. Usually it is also presumed that the function $f(x_1, x_2, \ldots, x_n)$ is indeterminate on the set $x_1 = \alpha_1$, $x_2 = \alpha_2, \ldots, x_n = \alpha_n$, if with the existence of the least root $y = \beta$ of the equation $g(\alpha_1, \alpha_2, \ldots, \alpha_n, y) = 0$ for at least one integral nonnegative value of $y = \gamma$ which satisfies the relation $0 \leq \gamma \leq \beta - 1$, the function $g(\alpha_1, \alpha_2, \ldots, \alpha_n, y)$ is indeterminate.

The arithmetic functions which can be constructed from the elementary arithmetic functions with the aid of the operations of superposition, primitive recursion and least root are termed partial recursive functions. If these functions are in addition everywhere determinate, then they are termed general recursive functions.

In this definition, just as in the definition of the primitive recursive functions, provision is made for the possibility of performing all admissible operations in any sequence and any finite number of times. There exists, however, the result of Kleene [41] which makes it possible to obtain any partially recursive function from two primitive recursive functions with the use of sequential application to them of a single least root operation and a single superposition operation. This result can be formulated more exactly as:

for any partial recursive function $f(x_1, \ldots, x_n)$ there exist two primitive recursive functions $g(x_1, \ldots, x_n, y)$ and $h(x)$ such that the function $f(x_1, \ldots, x_n)$ can be obtained from them in the form $f(x_1, \ldots, x_n) = h\,(\mu_y[g(x_1, \ldots, x_n, y) = 0])$, where μ_y is the least root operator. Here the function $h(x)$ can be chosen once and for all, regardless of the choice of f.

The partial recursive functions are the most common class of constructively definable arithmetic functions. They include, in particular, all the arithmetic functions which can be given in the form of finite recursive schemes of arbitrary form. By finite recursive scheme, here we understand any finite system of equalities $r = s$, where r and s are any finite (containing a finite number of symbols) expressions constructed from the known primitive recursive functions of unknown functions with numerical and literal arguments, where the values of the unknown functions for any given values of the arguments must be determined uniquely after a finite number of steps (depending on the selection of the values of the arguments) as a result of the application of two rules. The first rule (substitution rule) consists in the substitution into some one of the given equalities in place of one of the arguments some one of its numerical values. The second rule (replacement rule) makes it possible to use an equality of the form $x = f(x_1, x_2, \ldots, x_n)$, where $x, x_1, x_2, \ldots, x_n$ are numbers for the replacement by the quantity x of some occurrence of the quantity $f(x_1, x_2, \ldots, x_n)$ in one of the equalities $r = s$.

It is found that all the general recursive functions and only such functions can be represented in this manner. This situation makes it possible, following Erbran and Godel, to define the general recursive functions as functions represented by the finite recursive schemes of the form described above.

If, retaining the condition of single-valuedness, we do not require the definability of the values of the functions appearing in the scheme for all values of the arguments, we can represent the partial recursive functions by similar schemes. It is of essence that no recursive definitions (using finite schemes) make it possible to go beyond the limit of the class of partial recursive functions.

After accomplishing the numeration of the input and output words, any normal algorithm can be realized in the form of a partial recursive function. Conversely, any algorithm which is realizable with the aid of the partial recursive function is equivalent to some normal algorithm. Thus, we can draw the following important conclusion.

An algorithm is normalizable when and only when it can be realized with the aid of the partial recursive function.

This proposition shows that even on the basis of the arithmetic (numerical) approach to the theory of algorithms there is no departure from the class of the normalizable algorithms.

Let us consider two other approaches to the theory of algorithms proposed in 1936 by Post [63] and Turing [73].

In the algorithmic system proposed by Post, the input and output information is represented in the standard binary form, while the algorithm is in the form of a finite ordered set of rules termed orders. For the writing of the input, output and intermediate information use is made of a hypothetical endless <u>information tape</u> which is divided into individual cells, in each of which there can be located only a single letter (digit 0 or 1). Those cells in which ones are written are termed <u>signed</u> and those in which zeros are written are termed <u>unsigned</u>. At any instant of operation of the algorithm only a finite number of cells can be signed.

The operation of the algorithm is accomplished in discrete steps, in each of which there is performed one of the orders which constitute the algorithm. To each step there corresponds a definite <u>active</u> cell on the information tape. Some <u>initial</u> cell is fixed as the active cell for the first order. Further changes of the location of the active cell on the tape must be provided for in the algorithm itself. The orders which constitute the algorithm can belong to one of the following six types.

<u>First type</u>. Flag the active cell of the tape (write one in it) and go to the performance of the i-th order (i can be any number from the numbers used for the numeration of the orders of the algorithm).

<u>Second type</u>. Erase the flag of the active cell (write zero in it) and go to the performance of the i-th order.

<u>Third type</u>. Shift the active cell one step to the right and go to the performance of the i-th order.

<u>Fourth type</u>. Shift the active cell one step to the left and go to the performance of the i-th order.

Fifth type. If the active cell is signed (one is written there), then go to the performance of the j-th order, and if the active cell is not signed (zero written there), then go to the performance of the i-th order.

Sixth type. Stop, termination of operation of the algorithm.

Algorithms composed of any finite number of rules of the type described are called Post algorithms. It has been shown that the Post algorithms reduce to the algorithms realizable with the aid of the partial recursive function, and, conversely, any partial recursive function can be represented by an algorithm of the Post system. Thus, we can formulate the following proposition.

The class of all algorithms equivalent to the Post algorithms coincides with the class of all normalizable algorithms.

The algorithmic scheme proposed simultaneously by Post and Turing [73] is quite close to the scheme just described. In the Turing scheme, which is customarily termed the Turing machine, the information is also recorded on a bilaterally infinite information tape which is divided into individual cells. However, in contrast with the Post algorithm, here an arbitrary finite alphabet is required for the writing of the information. Each cell of the information tape serves for the writing of a single letter. This letter can be surveyed by a sensitive element, the so-called head of the Turing machine, which is capable of displacement along the information tape in both directions. The head of the Turing machine can be in a finite number of different states q_1, q_2, ..., q_n, can print in the surveyed cell any letter x_1, x_2, ...x_m and can shift to the right or left along the information tape by one cell.

The writing of the algorithm realized by the Turing machine is accomplished with the aid of the operating program of this machine, which is a set of five symbols of the form $x_i q_j x_k q_r s_p$. The written-out group of five symbols designates that the Turing machine head which is in the state q_j and senses the letter x_i recorded on the tape will print in place of this letter the new letter x_k (which can in a particular case coincide with the previously recorded letter x_i), transfers to the new state q_r (which also can coincide with the previous state) and makes a shift along the tape of the magnitude s_p, equal to ±1.

The original scheme of the Turing machine was intended for the writing out o. the values taken by an arbitrary single-place partial recursive function with values of the argument equal to 0, 1, 2, ... In this case, of course, the Turing machine must operate infinitely long. We can construct a Turing machine which computes the values of any a priori given partial recursive function. It is advisable, however, to modify the original scheme of the Turing machine described above. Let us assume that the last symbol s_p of the group of five symbols describing the operation of the Turing machine can take, in addition to the values ±1 introduced above, a third value--"stop machine". With this addition the Turing machine is converted into an ordinary algorithmic system. It either processes the input word p initially written on the tape infinitely long or after a finite number of transformation steps it stops. In the first case it is presumed, as usual, that the algorithm realized by the machine is not applicable to the input word p. In the second case the information remaining on the tape at the instant the machine stops is taken as the output word into which the machines transforms the given input word p, In this case, of course, it is necessary to have in the alphabet used for the recording of the information on the tape a special empty word to designate those cells in which no information is written.

We can show that all algorithms which are realizable with the aid of the described modifications of the Turing machines are normalizable and, conversely, any normalizable algorithm can be realized with the aid of a Turing machine specially constructed for this purpose. Making use of the sriting of the programs of operation of the Turing machines and of their input words in some standard alphabet, we can construct a universal Turing machine by exactly the same method used in constructing the universal normal algorithm (§2). Giving the universal Turing machine the representation of the program of any given Turing machine M and the representation of any input word p, we obtain the representation of the output word q into which the machine M transforms the input word p. If, though, the algorithm realized by the machine M is not applicable to the word p (the machine M works infinitely long on its transformation), then the algorithm realized by the universal Turing machine also is not applicable to the word formed from the representation of the word p and the program of the maching M.

Thus, in spite of the considerable qualitative difference, all the described algorithmic systems lead, in essence (with an accuracy to equivalency), to the same class of algorithms. This conclusion is still another confirmation that the modern theory of algorithms embraces an extremely broad class (if not all) of constructively definable alphabetic operators.

§5. THE CONCEPT OF ALGORITHMICALLY INSOLUBLE PROBLEMS

Every algorithm is the method of solution of some mass problem which can be formulated in the form of the processing not of one, but an entire set of input words into the corresponding output words. Since both the condition and the solution of any problem can be expressed in the form of individual words, every algorithm can be considered as a universal method for the solution of an entire class of problems.

A detailed analysis shows that there also exist those classes of problems for whose solution there is not and can not be a single _universal_ technique. The problems of the solution of this kind of problem are termed _algorithmicly insoluble problems_. However the algorithmic insolubility of the problem of the solution of problems of a particular class does not at all indicate the impossibility of the solution of any specific problem of this class. The question concerns the impossibility of the solution of _all_ problems of the given class by the _same_ technique.

For a better understanding of the problem of the algorithmic insolubility we shall present examples of algorithmicly soluble and algorithmicly insoluble problems.

A typical example of an algorithmicly soluble problem is that of the proof of identities in ordinary algebra. For simplicity we shall limit ourselves to the cases when the identities are constructed from rational numbers and letters (designated variables) with the aid of the addition, subtraction and multiplication operations. The following general technique for the solution of this problem is well dnown from the school algebra course: using the distributive way for multiplication, we remove the parentheses in the right and left sides of any given identity and perform the reduction of like terms in accordance with well known rules. After accomplishment of all these transformations, both the left and right sides of the original identity are transformed into polynomials. The identity will be valid when any only when these polynomials identically coincide with one another. In other words, the validity of the identity means that after the transfer of all the terms of the transformed identity into one side these terms mutually cancel, the result being the conversion of the identity into the trivial identity 0 = 0.

Thus, the identity problem in elementary algebra is algorithmicly solvable--there exists a single constructive technique which makes it possible after a finite number of steps to decide whether any fiven relation is an identity. We can, however, construct examples of such algebraic systems in which the identity problem is an algorithmicly insoluble problem. As such algebraic systems we might select, for example, the semigroups or groups given by systems of generating elements and defining relations. Examples of semigroups with insoluble identity problem were first found by Post [64] and corresponding examples for groups were found by Novikov [60].

Without writing out the defining relations explicitly, we shall clarify the essence of these examples. Let $x_1, x_2, \ldots, x_n$ be letters of some finite alphabet. The set of all words in this alphabet, including the empty word _e_, is termed a free _semigroup_ with the generating elements $x_1, x_2, \ldots, x_n$, if for the arbitrary pairs of words p, q there is introduced the multiplication operation amounting simply to the suffixing of one word to the other. We agree to designate the free semigroup with generating elements $x_1, x_2, \ldots, x_n$ by $F(x_1, x_2, \ldots, x_n)$, and the result of multiplying the word _p_ by the word _q_ we designate by pq.

In the free semigroup we can introduce any set of _defining relations_, which are formal equalities between two nonidentical words: $p_i = q_i$ ($i = 1, 2, \ldots$). Two words in the free semigroup F with the given system S of defining relations are termed identical, or mutually equivalent, if one of them can be obtained from the other by an arbitrary number of substitutions into the second word of the right sides of the defining relations in place of the left and, conversely, the left in place of the right. For example, in the semigroup with the system of generators (x, y) and one defining relation xy = yx the words

$p = xxy$ and $q = yxx$ are mutually identical since the first word can be obtained from the second as the result of two substitutions of the form described above: $q = yxx \rightarrow xyx \rightarrow xxy = p$. With the reverse substitution, the chain of substitutions written above can be read in the reverse direction, which makes possible not only the transformation of the word q into the word p but also of the word p into the word q.

The identity problem of words for the semigroups is formulated as follows.

Assume that in the arbitrary free semigroup F with a finite number of generators there is given any system of defining relations S consisting of a finite number of relations. We are required to find the single constructive technique which makes it possible after a finite number of steps to decide whether any two given words of the semigroup F with the system of defining relations S are identical or nonidentical.

For some systems of defining relations the problem formulated is solvable; however, as Post [64] has shown, there also exist such systems of defining relations for which the problem of the identity of the words is algorithmicly insoluble. This does not mean, of course, impossibility of establishing the identity or nonidentity of any fixed specific pair of words. There does not exist a single technique for the establishment of the identity of any pair of words, similar to the technique described above for the proof of the validity or nonvalidity of any relation in elementary algebra.

The problem of word identity for groups in its basic features coincides with the corresponding problem for the semigroups. The free group G with the generating elements $x_1, x_2, \ldots, x_n$ is constructed as the ensemble of words composed from the letters $x_1, x_2, \ldots, x_n$ and the "inverse" letters $x_1^{-1}, x_2^{-1}, \ldots, x_n^{-1}$. In this case two mutually inverse letters standing side by side cancel one another (become equivalent to an empty word)

$$x_i x_i^{-1} = x_i^{-1} x_i = e. \qquad (8)$$

In the determination of the identity of two words in a group with the system of defining relations S, we must take account not only of the relations appearing in this system but also the relations of the form (8). Just as for the semigroups, the word identity problem for groups which are specified by a finite number of generating and defining relations is algorithmicly insoluble in the general case. Examples of groups with insoluble word identity problem were first constructed by Novikov [60].

How can the algorithmic insolubility of a particular problem be proved? The classical example of such an insoluble problem is the problem of the recognition of the selfapplicability of algorithms. For the exact formulation of this problem we shall treat only normal algorithms in alphabets consisting of no less than two letters. With this assumption we can, without losing generality, stipulate that some letters of the alphabet of any algorithm with which we will be concerned will be identified with the two letters (0 and 1) of the standard binary alphabet. From the assumed condition, for any algorithm A considered, its representation A^u in the standard binary alphabet can be considered as the input word of this algorithm. If the word A^u appears in the domain of definition of the algorithm A, then the algorithm is termed <u>selfapplicable</u>, otherwise it is termed <u>nonselfapplicable</u>.

Both selfapplicable and nonselfapplicable algorithms exist. An example of the selfapplicable (normal) algorithm is the so-called <u>identity</u> algorithm in any alphabet $\mathfrak{X}$, which contains two or more than two letters. By definition this algorithm is applicable to any word <u>p</u> in the alphabet $\mathfrak{X}$ and transforms any input word into itself. An example of the nonselfapplicable algorithm is the so-called zero algorithm in any finite alphabet $\mathfrak{V}$. This algorithm is given by a scheme containing the identity substitution $\rightarrow y$ (where y is any letter of the alphabet $\mathfrak{V}$). By its very definition it is not applicable to any input word, and this means that it is not applicable ot its own representation.

The problem of the identification of the selfapplicability of the algorithms amounts to finding a single constructive technique which makes it possible, after a finite number of steps using the scheme of any given algorithm A in some fixed algorithmic system (for example, in the system of normal algorithms), to recognize whether the algorithm A is selfapplicable or not.

If we consider that the normalization principle formulated in §2 is valid, we can assume that the single constructive technique in question is none other than the normal algorithm B, defined on any word p, which is the representation of the arbitrary normal algorithm A and which transforms this word into two different fixed words q_1 and q_2 depending on whether the algorithm A is selfapplicable or not(the word q_1 is the code of the word "selfapplicable" and q_2 is the code of the word "nonselfapplicable").

On any input word l which is not the representation of any (normal) algorithm, the algorithm B also must be defined. Actually, otherwise, not obtaining any result after some number (sufficiently large) of steps of operation of the algorithm, we would not know whether the word l is the representation of a selfapplicable or nonselfapplicable algorithm. It is clear also that the result of the application of the algorithm B to any word which is not the representation of an algorithm must be different from the word q_1 and also from the word q_2.

Let us assume that the algorithm B with the indicated properties exists. In this case there exists the normal algorithm C in the same alphabet $\mathfrak{X}$, as the algorithm B, defined on all those and only those words in the alphabet $\mathfrak{X}$, which are the representations of nonselfapplicable algorithms (we recall that from the definition itself of the algorithm B, the alphabet $\mathfrak{X}$ includes in itself the standard binary alphabet).

Actually, let us construct the normal algorithm D in the alphabet $\mathfrak{X}$, whose domain of definition consists of only the single word q_2. Such an algorithm can be given, for example, in the form (normalized) of the superposition of two normal algorithms D_1 and D_2, the first of which is given by a scheme consisting of the single substitution $q_2 \rightarrow \cdot$, while the second is given by a scheme consisting of substitutions of the form $x_i \rightarrow x_i$, where x_i runs through all the letters of the alphabet $\mathfrak{X}$. It is clear that the first algorithm transforms into an empty word only the word q_2, while the domain of definition of the second algorithm consists only of an empty word. Therefore the domain of definition of the superposition D of the algorithms D_1 and D_2 will consist only of the word q_2, which we require.

After constructing the algorithm D, forming the superposition of it with the algorithm B, and normalizing this superposition, we arrive at the normal algorithm C in the alphabet $\mathfrak{X}$, whose domain of definition consists of all those and only those words in the alphabet $\mathfrak{X}$ which are forms of nonselfapplicable algorithms. However, this property of the algorithm C is intrinsically contradictory, since the algorithm C cannot be either applicable or nonapplicable to its own representation C^u.

Actually, in the first case the algorithm C would be applicable to its representation and therefore would be selfapplicable. But this would contradict the fact that as a result of its construction the algorithm C must be applicable only to the nonselfapplicable algorithms.

In the second case, being nonapplicable to its representation, the algorithm C would belong to the number of the nonselfapplicable algorithms. But then, by definition the algorithm C would have to be applicable to its representation, since it is applicable to the representation of all nonselfapplicable algorithms. Consequently, the algorithm C is selfapplicable.

Thus, the assumption on the algorithmic solvability of the problem of the recognition of selfapplicability leads to a logical contradiction and therefore is not valid, which proves the algorithmic undecidability of this problem.

We have substantiated this conclusion only for the condition that the algorithm normalization principle is valid. However, the nature of the contradiction used for the proof of the algorithmic insolvability of the problem of the recognition of the selfapplicability of algorithms is in actuality more profound. The reader who is familiar with the paradoxes of the theory of sets and of mathematical logic will easily note that this contradiction has the same nature as the contradiction in the known paradox of Russel which establishes the intrinsic contradiction of the concept of a "set of all sets not containing itself as an element."

This circumstance leads to the conclusion that the algorithmic undecidability of the problem of the recognition of selfapplicability is not a result of the narrowness of the modern exact concept of the algorithm. If we were able to construct an exact concept of the algorithm which includes certain nonnormalizable algorithms, then the problem of the recognition of the selfapplicability of the algorithms would remain as before algorithmicly undecidable.

From the algorithmic undecidability of the problem of the recognition of the selfapplicability of the algorithms, the algorithmic undecidability of a whole series of other problems is developed. The general method for these derivations amounts to the derivation from the assumption on the existence of the algorithm which solves a particular problem Q of the existence of the algorithm which solves the problem of the recognition of the selfapplicability of the algorithms. Since the latter is impossible, the existence of the algorithm which solves the problem Q also is impossible.

Using the genneral method, the algorithmic undecidability of a set of different problems has been proved, including the general problems of the identity of words for groups and semigroups considered above. We shall mention some other algorithmicly undecidable problems whose undecidability has been established by this same method. One problem is that of the <u>recognition of the applicability</u> of some algorithm to a particular word. There can be constructed an algorithm A, operating in some alphabet $\mathfrak{X}$, for which there does not exist an algorithm in the alphabet $\mathfrak{X}$, and in any expansion of it, which transforms into some fixed word those and only those words to which the algorithm A is not applicable.

The problem of the construction of an algorithm which transforms into the fixed word $\underline{p}$ all the words to which any given algorithm A is <u>applicable</u> is, as it is not difficult to see, algorithmicly undecidable; for its solution it is sufficient to construct the algorithm B which transforms into the word $\underline{p}$ all words in the alphabet of the algorithm A and to form the superposition of the algorithms A and B. We stipulate that an algorithm <u>annuls</u> particular words it it transforms them into the empty word $\underline{e}$. The problem of the <u>recognition of annulment</u> for any given algorithm A consists in the construction of the algorithm B (in the same alphabet as A) which annuls all those and only those words which algorithm A does not annul. This problem in the general case is algorithmicly undecidable, namely: we can select the algorithm A so that the algorithm B with the indicated properties cannot be constructed.

Quite frequently in the proof of the algorithmic insolvability of particular problems use is made of the Post [64] proof of the algorithmic insolvability of the following problem, which has been termed the Post combinatorial problem. Assume that in the arbitrary finite alpabet $\mathfrak{A}$ there are given any finite systems S of pairs of words $(p_1, q_1), \ldots, (p_n, q_n)$. We are required to construct a single constructive technique which will make it possible for any such system S after a finite number of steps to answer the question of whether we can construct a word $p_{i_1} p_{i_2} \ldots p_{i_k}$ from the first elements of the pairs of the system S such that it will coincide with the word $q_{i_1} q_{i_2} \ldots q_{i_k}$, constructed from the corresponding second elements of the same system of pairs.

The problem of matrix representability is also algorithmicly unsolvable. For the formulation of this problem we stipulate that a matrix is representable in terms of the matrix $U_1, U_2, \ldots, U_n$ if for some finite sequence (generally speaking with repetitions) $U_{i_1} U_{i_2} \ldots U_{i_k}$ of these matrices the product $U_{i_1} U_{i_2} \ldots U_{i_k}$ of all the matrices appearing in the given sequence coincide with the given original matrix U. The representation problem consists in finding the general constructive technique by which, after a finite number of steps for any matrix U and any finite system S of matrices, we would be able to know whether the matrix U is representable in terms of the matrices of the system S or not.

We recall that the algorithmic undecidability of all the indicated problems is proved on the assumption of the validity of the normalization principle; however, as noted above, the nature of this undecidability is more profound and, in a certain sense, is independent of this principle.

Manu-
script
Page
No.

[Footnotes]

13 The word p is termed the initial segment of the word q if the word q has the form $q = pr$, where r is any word (including, possibly, an empty word).

Chapter 2

BOOLEAN FUNCTIONS AND PROPOSITIONAL CALCULUS

§1. CONCEPT OF BOOLEAN FUNCTIONS

Boolean (or switching) function is the term customarily given to those functions for which all the arguments, and the functions themselves, can take on only two values.

The role of the boolean functions in cybernetics is determined by two basic characteristics. First, the boolean functions are a convenient apparatus for the description of the circuits of many information converters constructed using the discrete principle, since with current technology it is far easier to construct discrete elements functioning directly in the binary alphabet and not in some other alphabet. Second, the boolean functions are sidely used in mathematical logic, which is one of the foundations on which the automation of the complex thought processes is founded.

The use, along with the usual variables which take on numerical values, of the boolean variables, which have only two possible values, plays a significant role in the design of various kinds of practical algorithmic systems for programming on the electronic computers. The boolean functions can also be used successfully for the solution of certain general questions of the theory of algorithms, for example to refine the concept of algorithmic complexity. The two possible values of the variable which figure in the definition of the boolean functions can be designated arbitrarily. In practice, however, two notation systems are used most frequently. The first (for use of the boolean functions in the theory of automata circuits) assigns to the possible values of the boolean variables the notations 0 and 1. We shall term the symbols introduced, just as in the case of numerals, zero and one, considering that here the zero and one appear not as numerals, but only as convenient notations for the letters of the abstract binary alphabet. In the future we shall assign these symbols several properties which make it possible to consider them (with one exception) as ordinary numerals (this is precisely the convenience of the notation system being considered). But all such properties must be precisely defined before use. We cannot, in particular, yet make use of the properties of zero and one which result from the existence of the operations of addition and multiplication for numbers, since we have not yet defined these operations for these symbols.

In the second system of notation, the words "true" and "false" serve as the notations for the two possible values of the boolean variables. This system of notation is used in mathematical logic, primarily in the portion which is called propositional calculus. Its application is associated with the circumstance that in the propositional calculus the boolean variables are interpreted as the propositional variables, considered from the point of view of the truth or falsity of the proposition.

In the present and three following sections we shall make use of the first system of notation without specifying this each time. When it is necessary to make a transition from one system of notation to the other, we stipulate that one corresponds to true and the zero corresponds to false (we could, of course, assume exactly the opposite correspondence).

Let us consider the boolean functions of any finite number of arguments. Of the number of arguments is equal to _n_, then it is customary to term the corresponding function n-place. As a result of the fact that each boolean variable can take only two values, the domain of definition of any boolean function will of necessity be finite. It is easy to see that the domain of definition of an n-place boolean function can consist of a maximum of 2^n different elements, which are all possible sets of values of its n arguments. We will usually order the arguments of a given boolean function by assigning them the numbers 1, 2, ..., n. In this case the set of values of the arguments is identified with some cortege (finite ordered sequence) of zeros and ones. For example, the set of values $x_1 = 1$, $x_2 = 0$, $x_3 = 0$ of arguments of the three-place boolean function $f(x_1, x_2, x_3)$ can be abbreviated in the form of the cortege 100, and the set $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ can be written in the form of the cortege 001. In the future we frequently shall term these corteges simply sets (here the arguments are always numbered in a definite order--in the order in which they are encountered in the notation $f(x_1, x_2, \ldots, x_n)$ corresponding to the boolean function). The term boolean in application to a cortege (set) denotes that the corresponding cortege is composed of zeros and ones.

Each cortege of length n, composed of zeros and ones (a boolean cortege), can be identified with some vertex of an n-dimensional unit cube having the corresponding coordinates. For the two-dimensional case, when the n-dimensional cube reduces to a square, the method of identification of the boolean corteges with the vertices is shown in Fig. 4. As a result of the possibility of such identification, the boolean sets (corteges) will sometimes be termed points.

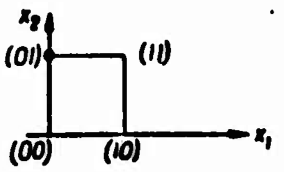

Fig. 4.

In the present chapter we shall limit ourselves (with the exception of specially stipulated cases) to the consideration of only those boolean functions whose domain of definition includes all sets of values of its arguments. Thus, the n-place boolean function must be defined at 2^n different points. If we do not exclude the case when a particular boolean function can be undefined on at least one of the sets, then it is termed a partial boolean function. The consideration of the partial boolean functions is useful for the synthesis of the circuits of descrete automata. In the theoretical aspect there is particular interest in the boolean functions which are everywhere defined, the more so since in case of necessity every partial boolean function can be redefined (generally speaking, by an arbitrary method) on those sets on which it was not initially defined. Therefore, speaking of the boolean functions hereafter (if not stipulated otherwise), we will understand them to be these everywhere-defined functions.

We remark also that in the consideration of a particular boolean function we shall consider the number of its arguments given. The necessity for this stipulation is due to the possibility of treating every n-place function as (n + 1)-place, (n + 2)-place, and in general as an (n + k)-place function for any natural number k. Actually, for example, the constant-function (equal identically to zero or one) can, if desired, be considered as a function of any number of arguments, arguments of which it is in actuality, however, independent. Similarly, we can to any function $f(x_1, x_2, \ldots, x_n)$ add any desired number of new arguments $x_{n+1}, \ldots, x_{n+k}$, on which the values of the function actually does not depend. For this it is sufficient to assume that for all sets of values of the variables $x_1, x_2, \ldots, x_{n+k}$ the following equality is valid

$$f(x_1, x_2, \ldots, x_n, x_{n+1}, \ldots, x_{n+k}) = f(x_1, x_2, \ldots, x_n).$$

We shall term the described operation of the conversion of the n-place function into an (n + k)-place function the operation of formal assignment of arguments. This operation is obviously applicable to any functions (and not only boolean).

As we noted above, there are exactly 2^n boolean sets (corteges) of length n. These sets can be considered as the representations of certain whole numbers in the binary number system such that the set $\alpha_1, \alpha_2, \ldots, \alpha_n$ is identified with the binary representation of the number $\alpha_1 \cdot 2^{n-1} + \alpha_2 \cdot 2^{n-2} + \ldots \alpha_{n-1} \cdot 2 + \alpha_n$ (here the boolean values 0 and 1 are considered simply as the usual numbers 0 and 1). We shall term this number the number of the corresponding set. The numbers of the sets vary from zero (for the set consisting only of zeros) to $2^n - 1$ (for the set consisting only of ones). The number of the set 010 will be the number $0 \cdot 2^2 + 1 \cdot 2 + 0 = 2$; the number of the set 101 will be the number $1 \cdot 2^2 + 0 \cdot 2 + 1 = 5$, etc.

Arranging the sets in columns one after the other in the order of increase of their numbers and placing alongside each set the value of the boolean function on this set, we obtain the value table of the boolean function. Since on each set the function can take either of two values (0 or 1) regardless of its values on the remaining sets, for m sets we can define exactly 2^m different (differing from one another by their values on at least one set) boolean functions. Keeping in mind the total number of sets for n variables (equal to 2^n) defined above, we come to the conclusion that the number of different boolean functions of n arguments, which we shall designate B(n), is determined by the equation

$$B(n) = 2^{2^n}. \quad (9)$$

With n = 1 the quantity B(n) is equal to 4, and with increase by 1 this quantity is squared: $B(n+1) = (B(n))^2$. Thus, if the number of single-place boolean functions is equal in all to 4, then the number of different two-place (boolean) functions will be equal to 16, three-place to 256, four-place to $256^2 \approx 65{,}000$, five-place to $256^4 \approx 4$

million, six-place to about 16 trillion ($16 \cdot 10^{18}$) and so on. The practical possibilities of sorting all the boolean functions are thus limited to the three-place or at best the four-place functions.

Although every boolean function can be given in the form of its value table, in the majority of cases of practical application of the theory of boolean functions this method of specification is inconvenient. Therefore, one of the primary tasks of our further constructions will be the development of new and more convenient methods of specifying the boolean functions. In this connection, of particular importance are the boolean functions of one and two arguments, since, as will be shown later, with their aid we can represent any boolean functions. Therefore, we shall make a more detailed study of the single-place and two-place boolean functions.

Of the four single-place functions $\varphi(x)$ which can in general be constructed, two functions are the constants 0 and 1 which are not explicitly dependent on x. Still another function simply repeats the value of its argument $\varphi(x) = x$ and therefore also is not of interest. The last, fourth, function, for which we introduce the special notation $\bar{x}$ or $\neg x$, always has a value which is the opposite to that of its argument: $\bar{0} = 1$ and $\bar{1} = 0$. This function is termed inversion or negation. The expression $\bar{x}$ (and also the expression $\neg x$) is read as "negation x" or "not x." In the theory of boolean functions, and also in the applications of this theory to the synthesis of automata circuits, following tradition, we shall make use of the notation $\bar{x}$. In mathematical logic (end of the present chapter and beginning of the sixth chapter) and also in the practical aspects of the theory of algorithms (end of the fifth chapter) it is for several considerations more convenient to use the notation $\neg x$.

Of the 16 different two-place boolean functions f(x, y) which in general can be constructed, six functions reduce to functions of a smaller number of arguments. These are, first, again the two constant functions (0 and 1), second, the two functions which repeat the values of some argument (x or y), and, third, two functions which are the negations of each of the arguments ($\bar{x}$ and $\bar{y}$).

The ten remaining functions $f(x, y)$, which actually depend on both of their arguments, can be divided into pairs such that the second function of the pair is the negation of the first function (i.e., it has on each set a value which is the opposite of the value of the first function). In this case use is actually made of the single-place boolean function $\bar{x}$ for the construction of the single-place negation operation on the set of all boolean functions. The application of the negation operation to any boolean function g can be treated as the substitution of the function in place of the argument of x into the function $\bar{x}$. Such a substitution of some boolean functions in place of the arguments of other boolean functions (termed superposition of these functions) will be widely used hereinafter for the formation of various operations on the set of boolean functions (boolean operations). For the designation of the operations thus constructed, we usually make use of the notation of the boolean functions which generated these operations. In our case $\bar{g}$ (or $\neg g$) will serve for the notation for the negation of the arbitrary boolean function g.

The separation described above of the two-place boolean functions into pairs $(g, \bar{g})$ makes it possible to actually limit ourselves to the description of only five functions, which we select as the first elements of the pairs indicated.

Let us begin the description with conjunction, also termed (logical) product, or the logical AND operation. In mathematical logic it is customary to designate the conjunction of the variables x and y by

x & y or $x \wedge y$ (we shall use the second of these notations). By definition, the conjunction $x \wedge y$ is equal to one when, and only when, both of its arguments $\underline{x}$ and $\underline{y}$ are equal to one.

For the conjunction $x \wedge y$ to be equal to zero it is sufficient that at least one of its arguments ($\underline{x}$ or $\underline{y}$) become zero. These properties of the conjunction are completely analogous to all the properties which the product xy would have if the cofactors composing it could take on only two <u>numerical</u> values--0 and 1. This circumstance suggests considering the <u>boolean constants</u> (0 and 1) as sort of "pseudo-numbers" for which the multiplication operation is defined which possesses all the properties of the usual multiplication operation for the <u>numbers</u> 0 and 1:

$$0 \cdot 0 = 0, \quad 0 \cdot 1 = 0, \quad 1 \cdot 0 = 0, \quad 1 \cdot 1 = 1.$$

In the theory of boolean functions and in its applications to the theory of automata, it is convenient to take precisely this point of view. Moreover, in these cases we shall simply identify the conjunction operation with multiplication, both in name and in form of representation. In other words, in place of the notation $x \wedge y$ we shall use the notation $x \cdot y$, or xy, and also shall make use of the terms "product," cofactor" and all the properties of multiplication from conventional elementary algebra. It is easy to understand that, as a result of the coincidence of the definitions, multiplication in our case will have all the general (satisfied identically) properties of multiplication in conventional algebra (commutativity, associativity, and so on). At the same time, the limitation on the set of possible values of the quantities leads to the appearance for the logical multiplication which we are considering of some properties which conventional multiplication does not have. For example, in the case of logical multiplication the identity relation $x \cdot x = x$ becomes invalid if in place of the values 0 and 1 we substitute into this relation other numerical values of the quantity x.

Just as in the case of negation, multiplication (conjunction) can be considered not only as a function, but also as an operation on the set of all boolean function. For this purpose it is sufficient in place of the independent variables x and y to substitute in the product xy two arbitrary boolean functions f and g: $p = fg$. Similarly, any other two-place boolean function $b(x, y)$ defines a two-place, or, as it is usually customary to say in algebra, binary operation on the set of all boolean functions, which we shall term and designate just the same as the corresponding function $b(x, y)$. Of course, in this case the independent variables x and y are replaced by the arbitrary boolean functions f and g. Hereafter we shall use the described technique for the introduction of new binary operations on the set of boolean functions without detailed explanations.

The possibility of the interpretation of conjunction as conventional multiplication suggests also looking for boolean analogs for conventional (numerical) addition. In contrast with multiplication, here there cannot be complete analogy, of course, since the equality $1 + 1 = 2$ in the case of conventional addition introduces a third quantity (two) which differs from both zero and one. With the limitation to only the boolean (binary) alphabet, the direct interpretation of this fact is, of course, impossible. Therefore we can define two different (but incomplete) analogs of numerical addition for the boolean quantities, setting the "sum" of two ones equal to either one or zero.

The operation (two-place boolean function) which arises with the first assumption is termed disjunction, logical addition, and also logical (the so-called inclusive) OR. For the designation of this operation we fix the special symbol (disjunction sign) $\vee$. Thus, the disjunction of the two quantities x and y (independent variables or functions) will be designated as $x \vee y$. The quantities x and y themselves in this case are termed the logical addends, or more frequently the disjunctive terms.

The system of relations which completely defines the operation of disjunction is written in the form $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$. The first three relations are exactly the same as in the case of conventional (numerical) addition, and only the fourth relation differentiates logical addition from conventional. In view of the relations introduced, the disjunction of the two quantities x and y is equal to zero when and only when both these quantities become zero. If even one of the quantities indicated takes the value 1, then this same value of 1 is taken by the disjunction itself, regardless of the value of the other disjunctive term.

A more fortuitous analog of conventional (numerical) addition is obtained in the case when the "sum" of the two ones is assumed to be equal to zero. The operation which arises in this case (two-place boolean function) is usually termed the non-equivalence operation, exclusive OR, and also modulo two addition. The last term is associated with the fact that this operation coincides with modulo two addition as defined in number theory if the zero and one are considered as ordinary numbers.

For brevity we shall term this operation simply addition and shall use such terms as sum and addend by analogy with conventional addition. We shall use the usual (+) sign to designate the operation of modulo two addition. In order to emphasize that we are not discussing conventional addition, we will at times put a circle around this symbol.

The operation of addition of boolean quantities is defined by the following four relations: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, $1 + 1 = 0$. The first three of them are exactly the same as in the case of conventional (numerical) addition (and the same as in the case of logical addition--disjunction), so that the specific nature of the operation introduced is defined primarily by the fourth relation. With this same relation there is associated the term for the addition operation, exclusive OR , which is used in mathematical logic. If we interpret one as true and zero as false, then the sum of two boolean quantities will be true when and only when <u>either</u> the first <u>or</u> second quantity is true, but not when they are <u>both</u> true. In the case of the logical sum (inclusive OR) the sum is also true when both addends (disjunctive terms) are true together. OR in this case does not exclude the simultaneous truth of both terms, it does <u>not separate</u> the question of the truth of the sum into two mutually exclusive cases, and this is the source of the association of the term "inclusive" as applied to OR in the logical sum (disjunction).

Still two more two-place boolean functions are the result of the single binary operation termed <u>implication</u>, or the <u>operation of logical succession</u>. We use the symbol $\supset$ for the designation of this operation. Implication is defined by the following four relations: $0 \supset 0 = 1$, $0 \supset 1 = 1$, $1 \supset 0 = 0$, $1 \supset 1 = 1$. In the implication $x \supset y$, in contrast with multiplication, disjunction and addition, the order in which the terms are arranged is of essential importance. With a reversal of this order the value of the implication changes so that $x \supset y$ and $y \supset x$ are two different boolean functions.

If we designate the two-place boolean function $f(x, y)$ by the cortege $\alpha_0\alpha_1\alpha_2\alpha_3)$, where α_i is the value taken by this function on the set with the number $i(i = 0,1,2,3)$, then the implication $x \supset y$ will correspond to the cortege (1101) while the implication $y \supset x$ corresponds to the cortege (1011). We note at the same time that the product, disjunction and sum of the variables x and y, regardless of the order in which these variables are written, are respectively the corteges: (0001), (0111) and (0110).

From a consideration of all the corteges presented, it follows, incidentally, that all five of the two-place boolean functions which we have defined (product, disjunction, sum and two implications) are pairwise different. It is easy to see that the cortege for the negation of any boolean function is obtained from the cortege for the function itself by replacing all the zeros by ones and all the ones by zeros. Using this rule, we can determine the cortege for negation of the product $\overline{xy}$, negation of the disjunction $\overline{x \vee y}$, negation of the sum $\overline{x + y}$, and the two negations for the implications $\overline{x \supset y}$ and $\overline{y \supset x}$. These corteges will be respectively (1110), (1000), (1001), (0010) and (0100).

It is easy to verify that, together with the five functions previously introduced, the five new functions (negations of the preceding five) compose a system of ten pairwise different two-place boolean functions. They all differ also from the constant-functions 0 and 1 and the functions x, y, $\bar{x}$, $\bar{y}$, considered as functions of the two variables x and y, since the latter functions are characterized by the corteges (0000), (1111), (0011), (0101), (1100), (1010) respectively. Thus, we have written out all 16 of the two-place boolean functions which can in general be constructed.

Let us make a few more remarks concerning the functions introduced above. The function $\overline{xy}$ (negation of the product) which is characterized by the cortege (1110) and the binary operation which is defined by it are customarily termed Sheffer's stroke function. It is easy to verify (using the definitions of negation and disjunction) that the Sheffer stroke can be represented not only in the form of the negation of the product $\overline{xy}$, but also in the form of the disjunction of the negations $\bar{x} \vee \bar{y}$.

The negation of the disjunction $\overline{x \vee y}$--the so-called Pierce function--characterized by the cortege (1000), can be represented also in the form of the product of the negations of the variables x and y, i.e., in the form $\bar{x} \cdot \bar{y}$. It is easy to see that both the Sheffer stroke and the Pierce function, similar to the product, disjunction and sum, are symmetric functions, i.e., they do not change their values with permutation of the arguments.

The negation of the sum $\overline{x + y}$, termed the equivalence operation or logical equivalence possesses a similar property. For the designation of this function and also for the binary operation defined by it, we use the special symbols $\sim$ or $\cong$ (equivalence symbol). The function $\overline{x + y} = x \sim y$ is characterized by the cortege (1001). The terms "equivalence" and "nonequivalence" as applied to the functions $x \sim y$ and $x + y$ respectively emphasize the fact that the first function is equal to one when and only when the values of its arguments are equal to one another, and the second--when the values of its arguments are unequal.

The function (binary operation) of implication can be expressed by disjunction and negation. It is easy to verify that $x \supset y = \bar{x} \vee y$ and $y \supset x = x \vee \bar{y}$. Negation of an implication, also termed the inhibit function, is easily expressed by the product and negation: $\overline{x \supset y} = x \cdot \bar{y}$, $\overline{y \supset x} = \bar{x} \cdot y$. Both implication and the inhibit function are examples of asymmetric boolean functions, since they change their values with permutation of the arguments.

In conclusion we note that in reading formulas the conjunction symbol $\wedge$ (or &) is pronounced as "and," the disjunction symbol $\vee$ is read as "or," the sum sign + (or $\oplus$) is read "plus," the implication sign $\supset$ is read "implies," the equivalence sign $\sim$ (or $\cong$) is read as equivalent," and the negation sign (or $\neg$) is read as "not."

All the ten listed two-place boolean functions correspond to the respective two-place boolean operations, which we shall designate and name exactly the same as the functions which define them.

§2. BOOLEAN ALGEBRA

Boolean algebra will be termed the set of all (finite-place) boolean functions considered together with the operations of negation, disjunction and multiplication (conjunction) specified on them.

We shall use the letters u, v, w, ... (with or without subscripts) to designate any elements of boolean algebra, i.e., in other words, any boolean functions. One of the primary problems of boolean algebra is the establishment of the identity relations of the form $A(u,v,w, \ldots) = B(u,v,w, \ldots)$ where $A(u,v,w, \ldots)$ and $B(u,v,w, \ldots)$ designate formulas, i.e., expressions of boolean algebra, constructed from a finite number of letters u,v,w, ..., the signs of the three operations of the algebra, the boolean constants (0 and 1) and parentheses for the designation of the order of performance of operations.

The formulas must be constructed properly. In other words, they must reduce to completely determinate boolean functions after the selection of particular boolean functions as values of the letters u,v,w, ... appearing in these formulas. We can give a rigorously formal definition of the properly constructed formula, introducible recurrently using the rule: all the letters u,v,w, ... (with or without subscripts) and the constants 0 and 1 are properly constructed formulas. If A and B are properly constructed furmulas, then $(\overline{A})$, $(A)\vee(B)$ and $(A)\cdot(B)$ are also properly constructed formulas. A set of properly constructed formulas is considered coincident with the set of all formulas which can be obtained as the result of sequential (multiple, generally speaking) application of this rule.

The introduction of each additional operation into the formula is accompanied by the appearance of one or two pairs of parentheses. To avoid excessive cumbersomeness of the formulas, we somewhat expand the concept of the rule for the construction of the formula, making it possible to drop some parentheses by analogy with the way this is done in elementary algebra. To do this we introduce the rule on the priority of operations: other conditions being equal, negations are performed first, then multiplication, then disjunction. When it is necessary to perform operations in a different order, parentheses are required. In addition, the negation sign written by a bar over an entire expression would have had to have been written. It will also be established later that the order in which like operations are performed which follow directly after one another in the formula is of no concern, so that in this case the parentheses are again redundant and can be dropped. Finally, we recall that the multiplication sign between letters can be dropped.

All the properly constructed formulas obtained as the result of the described expansions will hereafter be termed simply formulas, permitting using in them in addition to the letters u,v,w, ... any other letters of the Latin alphabet.

There is a very simple general rule for the verification of the correctness of the identity relations in boolean algebra. The essence of this rule amounts to the following.

Every formula A(u,v,w, ...) of boolean algebra can be considered as the representation of some boolean function of the variables u,v,w, ... Actually, of we assign these variables some constant values ($\bar{0}$ and 1) then, using the relations which define the operations of negation, disjunction and multiplication (i.e., relations of the form

$\bar{0} = 1$, $0 \vee 1 = 1$ and so on), we can after a finite number of steps find the value (0 or 1) of the formula A(u,v,w. ...) for the selected values of the variables u,v,w, ... and this then means that our formula is some everywhere-defined boolean function of the variables u,v,w, ...

It is easy to understand that the (identity) relation A(u,v,w, ...) = B(u,v,w, ...) is valid in and only in the case when the formulas A(u,v,w, ...) and B(u,v,w, ...) represent one and the same boolean function of the variables u,v,w, ... For the verification of the fact of the indicated equality of the two representations it is sufficient to verify whether the values of these representations on all sets of values of the variables u,v,w, ... coincide or do not coincide.

Thereby we have constructed a general algorithm, suitable of the verification of the correctness of any identity relations in a boolean algebra, since in view of the finiteness of the number of sets of values for any finite number of sets of the boolean variables the verification described always terminates after a finite number of steps.

Moreover, it becomes clear that it is sufficient to establish the identity relations in the boolean algebra for the case where all the letters appearing in these relations are considered as <u>independent</u> (boolean) variables. In case of necessity, moreover, any boolean functions can be substituted in place of these variables.

We shall designate the independent variables by the letters x, y, z (with or without subscripts). We shall also use these same letters for the writing of the identity relations of boolean algebra. We shall make a verification of the indicated relations with the aid of substituting into them all the possible sets of values of the variables (letters) appearing in these relations.

As an example let us consider the <u>commutativity relation for multiplication</u>

$$xy = yx. \tag{10}$$

To convince ourselves of the correctness of this relation it is sufficient to note that its left and right parts are equal to zero on the sets (00), (01), (10) and equal to one on the set (11). In view of the triviality of such a verification we shall not repeat it in the future and shall limit ourselves to simply writing out the relations we need, which we shall also term <u>laws</u> or <u>rules</u>.

In addition to the relation (law) of commutativity, for multiplication there also exist the so-called law of associativity, expressed by the equality

$$x(yz) = (xy)z. \tag{11}$$

Multiplication satisfies still another law, usually termed the <u>idempotency law</u>

$$xx = x. \tag{12}$$

As a result of this law, the concepts of power and raising to a power have no actual importance for the boolean algebra.

The laws of commutativity, associativity and idempotency extend also to the disjunction operation. The corresponding relations are written

$$x \vee y = y \vee x; \tag{13}$$

$$x \vee (y \vee z) = (x \vee y) \vee z; \tag{14}$$

$$x \vee x = x. \tag{15}$$

Multiplication and disjunction are related with one another by the first and second distributive laws, which can be expressed by the relations

$$x(y \vee z) = xy \vee xz; \tag{16}$$

$$x \vee yz = (x \vee y)\cdot(x \vee z). \tag{17}$$

We note that, on the strength of the agreements made on the priority of the operations, the right side of relation (16) is a simplification (as a result of discarding the redundant parentheses and the multiplication sign) of the formula $(x \cdot y) \vee (x \cdot z)$, while the left side of relation (17) is a simplification of the formula $(x) \vee (y \cdot z)$.

For multiplication and disjunction there are valid the so-called <u>absorption rules</u>, expressed by the following relations

$$x \vee xy = x; \tag{18}$$

$$x(x \vee y) = x. \tag{19}$$

For the negation operation the <u>law of double negation</u> is of great importance

$$\bar{\bar{x}} = x. \tag{20}$$

On the strength of this law any even number of negations performed in sequence does not alter the result, while any odd number is equivalent to performing a single negation.

For the various transformations in boolean algebra we frequently need to make use of the so-called <u>de Morgan rules</u>, which combine together all three algebraic operations,

$$\overline{xy} = \bar{x} \vee \bar{y}; \tag{21}$$

$$\overline{x \vee y} = \bar{x} \cdot \bar{y}. \tag{22}$$

We point out several more relations which include the constants 0 and 1:

$$x \vee \bar{x} = 1; \tag{23}$$

$$x\bar{x} = 0; \tag{24}$$

$$x \cdot 0 = 0; \tag{25}$$

$$x \cdot 1 = x; \tag{26}$$

$$x \vee 0 = x; \tag{27}$$

$$x \vee 1 = 1; \tag{28}$$

$$\bar{1} = 0; \tag{29}$$

$$\bar{0} = 1. \tag{30}$$

Relation (23) is called the <u>law of the excluded middle</u>, relation

(24) is the law of contradiction. Relations (25) and (28) can be considered as particular cases of the absorption rules.

Let us consider some corollaries from this system of relationships. From the laws of commutativity and associativity for disjunction and multiplication, there follows the possibility of performing in any order the actions for finding the values of the product and the disjunction for any finite number of terms. From this there follows the previously noted possibility of writing formulas of the form $x_1 \vee x_2 \vee \cdots \vee x_n$ and $x_1 x_2 \ldots x_m$ without parentheses with no chance of ambiguity as the result of variations of the order of performing the operations.

We note also that, as follows from relations (25) and (28), the presence of even a single one in the disjunction of the form $x_1 \vee x_2 \vee \ldots \vee x_n$ is sufficient to transform the entire disjunction into a one, just as the presence of even a single zero cofactor in the product $x_1 x_2 \ldots x_m$ transforms this entire product into zero. At the same time, relations (26) and (27) show that in any disjunction the terms equal to zero can be dropped, and in any product the terms equal to one can be dropped.

On the strength of relation (20), any number of negations performed in sequence reduces either to a single negation or in general to the absence of any negations. We shall use $\tilde{x}$ (read as "wavy x") to designate an expression which can be equal to either of the two expressions x or $\bar{x}$. Following the rule established above for the verification of identity relations in boolean algebra, we shall term the formulas representing the same boolean function of the variables appearing in them equal, or equivalent, to one another. Although the equality or inequality of any two formulas of boolean algebra can in principle be verified by means of the sorting of all possible combinations of the values of the variables appearing in them, with an increase of the number of variables this method becomes excessively cumbersome and is not suitable in practice. Therefore, one of the primary tasks of boolean algebra is the development of more economical methods of establishing the various kinds of relations which obtain in this algebra.

For the resolution of this problem we can make use of the previously derived relations (10)-(30), applying them repeatedly and in various combinations. For example, two-fold application of relation (12) makes it possible to establish the validity of the relation $xxx = x$, multiple application of relations (10) and (13) makes it possible to extend the laws of commutativity for disjunction and the product to any desired number of disjunctive terms and, correspondingly, cofactors. Thus, there arises the possibility of proving various relations in boolean algebra by transforming their left and right sides using relations (10)-(30). If in doing this we manage to reduce the left and right sides of some relation to the same formula, then the validity of the corresponding relation is thereby established.

It is not clear _a priori_ whether such a method makes it possible to derive _all_ the relations existing in boolean algebra. However, in actuality such derivation is always possible. To establish this fact, let us define some standard type of formula to which we shall try to reduce all the formulas of boolean algebra. In the reduction of a particular formula A of boolean algebra to the standard form we shall always fix some finite set M of the boolean variables $x_1, x_2, \ldots, x_n$, of necessity including all the variables which occur in the formula in question. We shall term every product of the variables or their negations (i.e., the product of the form $\tilde{x}_{i_1} \tilde{x}_{i_2} \ldots \tilde{x}_{i_k}$) an elementary product if each letter is encountered in the product no more than one time.

For example, the products $\bar{x}_1 x_2$ or $\bar{x}_1 x_2 x_3$ are elementary, while the products $x_1 \bar{x}_1$ or $x_3 \bar{x}_2 x_3$ are nonelementary. We shall include among the elementary products the variables x_i themselves and their negations $\bar{x}_i$, considering them as products consisting of a single cofactor. It is convenient also to consider that the constant 1 is an elementary product--the product of zero (empty set) cofactors. The number of cofactors in a product is called its _length_. The elementary products for a selected set M of variables can thus have any length from 0 to _n_ inclusive.

The elementary products of maximal length (in the present case, of length _n_) are customarily termed _constituents of unity_ for the selected set (M) of variables. It is easy to see that every constituent of unity contains all the variables of the set M (either in the direct form or in the form of the negation) precisely one time each, and that the total number of all such constituents is equal to 2^n.

The disjunction of any number of elementary products which does not contain two identical products is termed the _disjunctive normal form_. The disjunctive normal form which consists exclusively of constituents of unity is termed the _ideal disjunctive normal form_.

Just as in the case of the products, in this definition it is not excluded that the disjunction in question can consist of a single term (disjunction of length 1) and even of an empty set of terms (disjunction of length 0). In the latter case the disjunction is taken equal to zero by definition. Thus, the formulas $0,\ x_1,\ x_1 \vee x_2 \bar{x}_3,\ 1$ can be considered as disjunctive normal forms. The first of these formulas consists of an empty set of terms, the second consists of a single term, the third consists of two terms, and the fourth consists again of a single term which is the elementary product of zero length.

Replacing in all the definition the disjunctions by products,

products by disjunctions, the (boolean) constant 0 by the (boolean) constant 1 and vice versa, we obtain respectively the definitions of the elementary disjunctions, constituents of zero, the conjunctive normal form and the ideal conjunctive normal form.

In boolean algebra, as a result of the fact that with replacement of zero by one and one by zero the disjunction is transformed into conjunction and vice versa, there arises a unique duality of the properties of disjunction and conjunction (multiplication). Performing such a replacement, we can automatically for any property (relation) derived herafter obtain its dual property (relation). In particular, to all the properties of the disjunctive normal forms we can associate, using the indicated duality law, the corresponding properties of the conjunctive normal forms. Since this association is accomplished each time almost automatically, we shall limit ourselves in the future to the consideration of only the disjunctive normal forms.

Using relations (10), (11), (13)-(16), (23) and (26), we can transform any disjunctive normal form into its equivalent ideal disjunctive normal form. Let us consider the process of such a transformation using the example of the disjunctive normal form of three variables $x \vee \bar{y}z \vee \bar{x}yz$, which for brevity we shall designate with the single letter f.

The third term of this formula is a constituent of unity and therefore does not require any transformations. In order to be a constituent of unity, the second term lacks the multiplier $\tilde{x}$ (i.e., x or $\bar{x}$), and the first term lacks the factors $\tilde{y}$ and $\tilde{z}$. On the basis of relations (23), (26) we can write that $f = x(y \vee \bar{y})(z \vee \bar{z}) \vee \bar{y}z(x \vee \bar{x}) \vee \bar{x}yz$. Using the first distributive law (relation (16)) and relations (10), (11), (13)-(15) we sequentially bring our form to the form $f = (xy \vee x\bar{y})(z \vee \bar{z}) \vee \bar{y}zx \vee \bar{y}z\bar{x} \vee \bar{x}yz = xyz \vee x\bar{y}z \vee xy\bar{z} \vee x\bar{y}\bar{z} \vee x\bar{y}z \vee \bar{x}\bar{y}z \vee \bar{x}yz = xyz \vee xy\bar{z} \vee x\bar{y}z \vee x\bar{y}\bar{z} \vee \bar{x}\bar{y}z \vee \bar{x}yz$. The last expression in this chain of equalities is the desired ideal disjunctive normal form. We now establish the following important result.

Theorem 1. With the aid of relations (10)-(30) any formula of boolean algebra can be reduced to the ideal disjunctive normal form.

Actually, using several times the de Morgan rules (21) and (22), the double negation law (20), and also the relations (29) and (30), any formula $A(x_1, x_2, \dots x_n)$ of boolean algebra can be reduced without difficulty to its equivalent formula $B(x_1, x_2, \dots, x_n, \bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$, which does not contain any negations other than the negations associated directly with the letters $x_1, x_2, \dots, x_n$. It is easy to clarify the transformations required in this case from the example

$$\overline{x \vee \bar{y}z \vee y\bar{z}} = \overline{(x \vee \bar{y}z)} \cdot \overline{(y\bar{z})} = \bar{x}\,\overline{(\bar{y}z)}\,(\bar{y} \vee z) = \bar{x}\,(y \vee \bar{z})(\bar{y} \vee z).$$

The described technique of sequential dropping of the negation signs is applicable to any formula of boolean algebra.

The formula $B(x_1, x_2, \dots, x_n\ \bar{x}_1, \bar{x}_2, \dots, \bar{x}_n)$ is constructed from the letters (with or sithout negations) shown in its designation with the use of only the multiplication and disjunction operations. Relations (10), (11), (13), (14) and (16) show that expressions, just exactly as in the usual school algebra course (considering disjunction as addition), can be transformed to remove all the parentheses and to group all like terms. After such transformation with subsequent account for relations (25), (26) and (27) our formula is transformed into a disjunction of certain products of the letters $x_1, x_2, \dots, x_n$ and their negations. With the aid of relations (10), (12), (24) and (25) all these products can be transformed to their equivalent elementary products or zeros. Now, using formulas (27) and (15), we reduce our formula to the ideal disjunctive normal form. An example of this was discussed above. Thereby the theorem is completely proved.

It is clear that the resulting ideal disjunctive normal form is equivalent to the original formula since we used equivalent transformations in each of the steps described above.

We note that all the steps performed are reversible, so that with the use of relations (10)-(30) we can also accomplish the reverse conversion from the constructed ideal disjunctive normal form to the original formula $A(x_1, x_2, \ldots, x_n)$.

Theorem 2. For the arbitrary boolean function $\underline{f}$ of any finite number of variables $x_1, x_2, \ldots, x_n$ there can be constructed one and, with an accuracy to permutation of the disjunctive terms and cofactors, only one ideal disjunctive form with the same set of variables to which it is equal.

To each set $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of values of the variables $x_1, x_2, \ldots, x_n$ there corresponds exactly one constituent of unity $\tilde{x}_1 \tilde{x}_2 \ldots \tilde{x}_n$, which becomes unity on this set. This constituent is uniquely defined by the condition $\tilde{x}_i = x_i$, if $\alpha_i = 1$ and by $\tilde{x}_i = \bar{x}_i$ if $\alpha_i = 0 (i = 1, 2, \ldots, n)$. All the remaining constituents for the given set of values of the variables have zero values. Since in a disjunction the terms which are equal to zero can be discarded, then it becomes clear that the disjunction $\underline{g}$ of the constituents of unity corresponding to all the sets on which the values of the function $\underline{f}$ are equal to unity is an ideal disjunctive normal form equal (as a boolean function) to the function $\underline{f}$. It is clear also that every variation in the composition of the constituents of unity occurring in the form $\underline{g}$ will inevitably alter its value table and, naturally, will destroy the established equality. Consequently, the form $\underline{g}$ is defined uniquely by the function $\underline{f}$, Q.E.D.

In view of the indicated uniqueness of the definition, the form $\underline{g}$ is customarily termed the ideal disjunctive normal form of the considered function $\underline{f}$.

Two other important results follow directly from theorems 1 and 2.

<u>Theorem 3</u>. Any boolean function can be represented in the form of a formula of boolean algebra.

<u>Theorem 4</u>. With the aid of relations (10)-(30) every formula of boolean algebra can be represented in any other formula which is equivalent to it (i.e., representing the same boolean function).

Actually, as the formula representing any given boolean function <u>f</u> we can choose its ideal disjunctive normal form. We can always transform any formula A into its equivalent formula B by means of the ideal disjunctive normal form <u>g</u>, which, as a result of theorem 2, will be common for formulas A and B. The chain of transformations which transforms the formula A into <u>g</u>, and the chain reducing B to <u>g</u> taken in reverse order (on the strength of theorem 1 such chains exist) constitute a chain of transformations which transform the formula A into the formula B.

We note that not all the relations (10)-(30) written out above from the proof of theorem 1 are used in the transformations (for example, relation (17) is not used). Therefore, if desired the system of relations (10)-(30) can be abbreviated such that theorems 2 and 4 will be valid as before.

The second remark concerns the fact that the method of transforming the formula A into its equivalent formula B by means of the ideal disjunctive normal form <u>g</u> common to both of them was necessary only to establish the principle of the possibility of conversion from A to B. In practice this method usually turns out to be too cumbersome, as a result of which we generally look for more direct ways to convert from A to B (although, of course, sometimes there may not be a way which is significantly shorter to get from A to B than the "roundabout" method indicated above).

An important problem which is solvable within the framework of boolean algebra is the problem of the minimization of formulas. The sense of this problem is the finding of a general technique (algorithm) which makes it possible for any formula of boolean algebra to find its equivalent formula having the minimal possible complexity.

As the criterion of the complexity of a formula it is most natural to take the number of operations appearing in this formula, so that, for example, the complexity of the formula $\bar{x}$ will be the number 1, while the complexity of the formula $(\bar{x} \vee y)(\bar{y} \vee z)$ will be the number 5 (two negations, two disjunctions and one multiplication). However, following the tradition established in the majority of the works on the minimization problem, we shall make use of a different criterion, taking the complexity of a formula to be the total number of letters appearing in it. Here we are speaking of the number of occurrences of the letters (including, possibly, identical letters in this number) and not of the number of different letters in the formula. Thus, for instance, in view of the criterion we have defined, the complexity of the formula $(x \vee y)(x \vee \bar{y})$ should be considered 4, not 2.

It is not difficult to understand that the set M(A) of the different formulas of boolean algebra whose complexity does not exceed the complexity of any fixed formula A will inevitably be finite. Therefore the problem formulated above of the minimization of formulas can in principle be resolved by the sorting of all the formulas of the set M(A) in the order of increasing complexity until a formula is found which is equivalent to formula A. However, the algorithm based on this sorting is so cumbersome that is is not suitable in practice.

The problem of the construction of more economical algorithms for the minimization of formulas in boolean algebra has not yet been solved in the general form. Therefore, in practice we limit ourselves, as a rule, to the problem of finding the minimal formula in a particular class of formulas and first of all in the class of all disjunctive normal forms. This problem is usually termed the problem of the minimization of the boolean functions, which, of course, is not entirely accurate, since we are not speaking of the minimization of the function (which remains unchanged in the minimization process) but of the minimization of the formulas which represent the function (in the present case--the disjunctive normal forms).

All the methods of minimization in the class of the disjunctive normal forms are based on the concept of the prime implicant. The implicant of the boolean function f is the term given to every boolean bunction g whose reduction to unity is possible only on those sets of values of the variables on which the function f reduces to unity. We stipulate that the implicant g covers with its unities some unities of the function f. From the properties of the disjunction it follows that the disjunction of any (finite) set of implicants $g_1, g_2, \dots, g_n$ of the function f will again be its implicant. If in this case the unities of the implicants $g_1, g_2, \dots, g_n$, considered all together, cover all the unities of the function f, then this disjunction simply coincides with the function f: $g_1 \vee g_2 \vee \dots \vee g_n = f$.

The reverse is also clear: any term of the disjunction coinciding with the function f is the implicant of this function f is the implicant of this function, and the unities of all the terms of the indicated disjunction all together cover all the unities of the function f. In particular, every disjunctive normal form g of the boolean function f can be considered as the covering of this function by the set of all terms of form g, each of which is the implicant of the function f. In this case the elementary products appear in the role of implicants.

We note that with a reduction of the length of the elementary product (as the rusult of dropping part of the cofactors) the number of unities covered by it is increased. The elementary product of maximal length (constituent of unity) for _n_ variables reduced to unity only at one point, while the elementary product of length $n - k$ reduces to unity at 2^k points. Therefore it is of advantage to cover any given function _f_ by elementary products of the minimal possible length, i.e., by such elementary products that they themselves are implicants of the function _f_, but none of their internal parts are implicants of this function. Such elementary products are customarily termed _prime implicants_ of the boolean function in question.

The set of all prime implicants of any boolean function _f_ covers all its ones. Therefore the disjunction _g_ of all prime implicants of the function _f_, termed the _reduced disjunctive normal form_ of this function. However, this representation will usually not be the most economical, since some prime implicants can civer ones which are already covered by the remaining implicants. Discarding from the form _g_ all such redundant implicants, we transform it into the so-called _irreducible disjunctive normal form_ of the function _f_ in question.

A boolean function can have, generally speaking, not one but several irreducible disjunctive normal forms. For instance, the function of the three variables x, y, z, reducing to zero only on the sets (000) and (111) and equal to unity on all the remaining sets, has five different irreducible disjunctive normal forms. At the same time, we can show that any two-place boolean function has a single irreducible disjunctive normal form.

It is easy to understant that among the irreducible disjunctive normal forms of any boolean function _f_ there are inevitably contained all its _minimal_ disjunctive normal forms (there may be several of them), i.e., those disjunctive normal forms of the function _f_ which contain the smallest number of letters in comparison with all the remaining disjunctive normal forms of this function.

We can construct sufficiently economical algorithms for finding all the prime implicants and all the irreducible disjunctive normal forms of any boolean function. However, for separating the minimal forms from the number of irreducible disjunctive normal forms there is not in the general case any significantly simpler method than the method of sequential sorting and comparison of all the irreducible disjunctive normal forms (see Zhuravlev [36]).

One of the most effective algorithms for finding the prime implicants and the irreducible disjunctive normal forms is the algorithm proposed by Blake [8]. The essence of the Blake algorithm is the following. It is not difficult to establish that in boolean algebra there is satisfied the identity relation of the form

$$AB \vee \bar{A}C = AB \vee \bar{A}C \vee BC. \qquad (31)$$

If in this relation we consider A to be a letter and B and C to be elementary products, then from relation (31) there is derived the rule for identity transformation of the disjunctive normal forms which makes it possible if they contain two terms of the form xp and $\bar{x}$q to complement them with a new term (elementary product) pq. It is possible, it is true, that this term vanishes or coincides with one of the disjunctive forms present in the form already. It is easy to understand that in view of the finiteness of the total number of elementary products (given variables) new terms will not be obtained by means of a finite number of steps of application of the indicated rule. Blake's result amounts to the statement that the transformed form of <u>f</u> after reaching suitable "stabilization" will contain all the prime implicants of the boolean function which it represents.

After obtaining the disjunctive normal form <u>g</u> containing all its prime implicants, it is not difficult to free it of all the terms which are not prime implicants. Actually, if any elementary product P from g is not a prime implicant, then, being in any case an implicant of the function g, it includes in itself some prime implicant p of this function and, consequently, can be represented in the form $P = pq$. Since in g there is the disjunctive term p, then it can be used to exclude from g the term $P = pq$ with the aid of the relation (18): $p \vee pq = p$. Such an exclusion is usually termed the elementary absorption operation. Its application to the disjunctive normal form g provides after a finite number of steps the cancellation of all the terms which are not prime implicantsand the conversion, thusly, of the form g into the simplified disjunctive normal form g_0.

In order to go from the form g_0 to some irreducible disjunctive normal form, we can find the redundant terms in g_0 by the same method of Blake: if some term in the form g_0 (or in any other disjunctive normal form consisting of prime implicants) can be obtained from the remaining terms with the aid of the application (possibly more than once) of relation (18), then this term is redundant and it can be excluded.

By applying such an exclusion process repeatedly, we reduce the form g_0 to the irreducible disjunctive normal form g_1. Actually, on the strength of the result of Blake presented above, with the aid of relation (30) we can obtain from the form g_1 all the prime implicants appearing in g_0. But further exclusion of terms of the form g_1 is not possible. Actually, if we attempt such an exclusion at least one of the unities of the function g_1 will be uncovered. It is clear that the prime implicant (excluded from g_1) covering this unity now cannot be recovered from the disjunction of the remaining terms by any elementary transformations, in particular with the aid of the application of identity (31).

In order to obtain all the irreducible disjunctive normal forms, the described exclusion method should be applied several times with variation of the order in which the attempts are made to exclude the various terms. As we mentioned above, the finding of the minimal disjunctive normal forms requires a complicated operation in the sorting of all the irreducible disjunctive normal forms (which can be of varied complexity). Therefore in practice the solution of the problem of minimization is usually limited to finding some one, randomly selected, irreducible disjunctive normal form.

As an example of the application of the Blake algorithm, we shall show the process of minimization of the disjunctive normal form $f=\bar{x}yz\vee$ $\vee xy\vee x\bar{y}\bar{z}$.

Applying relation (31) to the pairs composed from the first term with the second and from the first term with the third, we reduce the given form f to the form $f_1=\bar{x}yz\vee xy\vee x\bar{y}\bar{z}\vee yz\vee x\bar{z}$. Application of relation (31) to any pair of terms of the form f_1 does not lead to the appearance of new terms. Consequently, all the prime implicants of the function f (the function represented by the form f) are contained in the form f_1.

The application of the operation of elementary absorption to the form f_1 leads to the reduced disjunctive normal form $f_2 = xy\vee yz\vee x\bar{z}$. The first term of the form f_2 can be obtained with the aid of relation (31) from the remaining two terms and is thus redundant. Excluding it, we come to the form $f_3 = yz\vee x\bar{z}$, which does not contain redundant terms and which is, consequently, the desired irreducible disjunctive normal form. In the present case the irreducible disjunctive normal form is the only one and as the result of this it coincides with the minimal disjunctive normal form.

More detailed information on the various methods of minimization of the formulas of boolean algebra can be obtained in special monographs on the theory of the synthesis of the circuits of discrete automata (see, for example, Glushkov [26]). Some additional information on this question is presented also in §4 of the present chapter.

§3. THE CONCEPT OF COMPLETE SETS OF BOOLEAN OPERATIONS

Theorem 3 of the preceding section shows that for the representation of any boolean function in the form of a formula constructed from the arguments and the boolean constants 0 and 1, it is sufficient to use in all three types of boolean operations, negation, multiplication and disjunction. Every set of boolean operations which possess such a property is customarily called a complete set.

In addition to the set consisting of the operations of negation, multiplication and disjunction, we can also construct other complete sets of boolean operations. From the de Morgan relations (21) and (22) written out in the preceding section, it follows that the disjunction operation can be represented by the operations of negation and multiplication, and that the multiplication operation can be represented by the operations of negation and disjunction. Therefore a complete set of boolean operations can be composed from the negation operation and any of the two remaining operations of boolean algebra (multiplication or disjunction).

From the operations of any complete set of boolean operations there can be constructed any boolean operations, in particular the operations of negation, disjunction and multiplication. In order to perform the required construction it is obviously sufficient, using the operations of the complete set being considered, to represent the boolean function which defines the required operation. Conversely, if from the operations of some set we can construct the operations of negation and multiplication or negation and disjunction, then, in view of what we have said above, this is sufficient for the possibility of representing any boolean function and, consequently, for the completeness of our set. As a result we come to the following proposition.

Theorem 1. For the completeness of any set of boolean operations it is necessary and sufficient that with the aid of the operations of this set we can construct the function $\bar{x}$ and one of the functions xy or $x \vee y$.

Using the criteria of completeness from Theorem 1, we can relatively easily establish the completeness of many sets of boolean operations. One such set is, in particular, the set consisting of the operations of multiplication and addition (modulo two). Actually, it is easy to verify that the following relation is valid

$$\bar{x} = x + 1. \tag{32}$$

Thus, negation can be expressed by addition. Since multiplication itself appears in the set in question, on the basis of Theorem 1 we arrive at the conclusion on the completeness of this set.

With the aid of the operations of addition and multiplication there is constructed still another interesting algebra of the boolean functions, termed the Zhegalkin algebra. In its general properties (expressed by the identity relations) this algebra approaches most closely the algebra with the conventional addition and multiplication operations which is studied in high school. Like conventional addition, modulo two addition satisfies the associativity and commutativity relations (for boolean multiplication these properties were established in the preceding section). The distributive law $x(y + z) = xy + xz$ is also satisfied, making it possible to remove parentheses in expressions just as in conventional algebra.

After removal of the parentheses, any formula in the Zhegalkin algebra is represented in the form of the sum of the products of the variables, including, possibly, the products consisting of a single cofactor (single letters) and of a zero cofactor (the constant 1). On the basis of the relation xx = x and the commutativity of multiplication, we can consider that in any of the products obtained no letter will occur more than one time.

Identical products, just as in conventional algebra, are considered similar terms and are subject to the operation of reduction of like terms. The rules for this reduction are different from the corresponding rules in conventional algebra, amounting, in the final analysis, to the easily verifiable identity relation

$$x + x = 0. \tag{33}$$

Thus, any even number of identical addends mutually cancel, while any odd number is equivalent to only a single addend, since the zero addends do not alter the values of the sum and can be immediately stricken from the sum.

The reduction of likes terminates our description of the reduction process, which we shall call the process of reduction of formulas in the Zhegalkin algebra to the <u>canonical form</u>. Let us demonstrate this process using an example. Let there be given some formula f = = (x + y)(x + z) + y(z + x) of the Zhegalkin algebra. After removal of the parentheses, this formula takes the form f_1 = x + xy + xz + yz + + yz + xy. After combining like terms, the terms xy and yz, encountered twice in the formula, cancel and the formula itself is reduced to the final (canonical) form f_2 = x + xz.

In view of the completeness of the set of operations of the Zhegalkin algebra and the possibility of reduction of any of its formulas to the canonical form, every boolean function <u>f</u> can be represented in the Zhegalkin algebra by a formula of canonical form. It is not difficult to show that the last formula is determined uniquely by the function <u>f</u> with an accuracy to possible permutation of addends and cofactors. We shall call this formula the canonical polynomial of the given boolean function _f_.

The uniqueness of the determination of the canonical polynomial can be established by simple reasoning. Let f_1 and f_2 be two different canonical polynomials of the boolean function _f_. Being equal to this function, the polynomials f_1 and f_2 are equal to one another as functions (for all values of the variables). In the equality $f_1 = f_2$ identical terms in the right and left sides can be mutually cancelled. In the right and left sides of the identity relation arising after this $f_1' = f_2'$ there is not a single pair of identical terms (addends).

Let us fix one of the addends _p_ which is composed of the smallest number of letters in comparison with the remaining addends. Then all the remaining addends will differ from the selected addend _p_ by at least one letter. Let us fix the set of values of the variables so that all the letters appearing in _p_ take the value 1 and all the remaining letters take the value 0. On the strength of this remark, only one of the addends, and precisely the addend _p_, will become unity with the selected set of values of the variables, all the remaining addends will be equal to zero. But then the relation $f_1' = f_2'$ is brought to the form 1 = 0 (or 0 = 1), which is not possible of the original relation $f_1 = f_2$ was identical. Thereby we have refuted the assumption made in the beginning of our discussion on the existence of two different (although equal to one another as functions) canonical polynomials f_1 and f_2 for the same boolean function _f_.

Canonical polynomials which do not contain products of two or more variables (i.e., polynomials which are the sum of individual letters and, possibly, the constant 1, and also the polynomial identically equal to zero) are the so-called linear boolean functions. All the remaining boolean functions are nonlinear. Corresponding to this division of the functions, all the boolean operations which they determine are also divided into linear and nonlinear operations.

It is easy to see that with any superpositions (substitutions of one in the other) of linear boolean functions the functions resulting from the superposition will again be linear. This means, clearly, that with the aid of the linear (boolean) operations we cannot construct any nonlinear operation. This implies that every complete set of boolean operations must include at least one nonlinear operation.

The operations of negation and addition (modulo two) are linear operations, since the canonical polynomials representing their boolean functions will be the linear formulas $x + 1$ and $x + y$. At the same time the functions xy and $x \vee y$, and consequently the multiplication and disjunction operations which they define, are nonlinear. The first of them has as its canonical polynomial the formula xy, and the second--the formula $x + y + xy$. Both these formulas contain the nonlinear term xy.

Let us introduce still another division of the boolean functions and their corresponding boolean operations into two classes: the class of <u>monotone functions</u> (operations) and the class of nonmonotone functions (operations). To do this let us define for the sets of values of the boolean variables the order relation $\leq$, assuming that $0 \leq 0$, $0 \leq 1$, $1 \leq 1$ and that for any two sets of identical length $(\alpha_1\alpha_2 \ldots \alpha_n)$ and $(\beta_1\beta_2 \ldots \beta_n)$ the relation $(\alpha_1\alpha_2 \ldots \alpha_n) \leq (\beta_1\beta_2 \ldots \beta_n)$ is valid when and only when for <u>all</u> $i = 1, 2, \ldots, n$ $\alpha_i \leq \beta_i$. If these sets are different, then we shall say that the first of them is the <u>smaller</u> and the second is the <u>larger</u>. We note that certain sets, for example the sets (01) and (10), will in this case be <u>incomparable</u> with one another, since the definition presented does not make it possible to consider that one of them is larger or smaller than the other.

The boolean function f is termed monotonic if with transition from any smaller set A of values of its variables to any larger (in comparison with A) set B the value of the function cannot diminish, i.e., transition from the value 1 on the set A to the value 0 on the set B. If, however, even for one pair of the sets A, B such that $A \leq B$, $f(A) = 1$, and $f(B) = 0$, then the function f is termed a nonmonotonic boolean function. The boolean operations determined by these functions are correspondingly divided into monotonic and nonmonotonic.

For any superpositions (substitutions of the function into function) of the monotone boolean functions we again obtain monotone functions. Actually, if the functions $f(y_1, y_2, \ldots, y_m)$ and $\varphi(x_1, x_2, \ldots x_n)$ are monotone, and the function φ is substituted, say, in place of the variable y_1, then, on the strength of the nomotonicity of the function φ with any increase of the set of values of the variables y_2, y_3, ... y_m, x_1, x_2, ..., x_n, the set of values of the variables φ, y_2, y_3, ... y_m will either increase or remain unchanged. In both cases the value of the complex function $f(\varphi, y_2, y_3, \ldots, y_m)$, in view of the monotonicy of the function $f(y_1, y_2, \ldots, y_m)$, cannot diminish, which proves its monotonicity.

With transfer over to operations, the fact just established means that with the aid of only the monotone boolean operations we cannot construct any nonmonotone boolean operation (for example, negation), which implies that every complete set of boolean operations must of necessity include at least one nonmonotone operation.

The simplest example of nonmonotone operation is that of negation. It is found also, that in a certain sense every nonmonotone operation includes the negation operation. More precisely, the following proposition is valid.

Theorem 2. The negation operation can be constructed with the aid of any nonmonotone boolean operation.

Let us consider an arbitrary nonmonotone boolean operation. It is defined by some nonmonotone boolean function $f(x_1, x_2, \ldots x_n)$. In view of the nonmonotonicity of the function f, there are two sets A and B of values of its variables such that A is smaller than B, and the function f takes on the value 1 on the set A and the value 0 on the set B. The set A differs from set B in that in certain of the locations where in the set B there stand ones, in the set A there stand zeros. Replacing sequentially, one by one, these zeros by ones, sooner or later we arrive from the set A, where $f(A) = 1$, at set B, where $f(B) = 0$. Consequently, in one of the sequential replacements of zero by one the value of the function f must change from 1 to 0. This means that for some i $(1 \leq i \leq n)$ $f(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n) = 1$ and $f(\alpha_1, \alpha_2, \ldots, 1, \alpha_{i-1}, \ldots, \alpha_n) = 0$, where $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$ are certain boolean constants (0 or 1). But them, as it is easy to see, the boolean function $f(\alpha_1, \alpha_2, \ldots, \alpha_{i-1}, x, \alpha_{i+1}, \ldots, \alpha_n)$ of the one variable x can be nothing other than the negation of this variable, i.e., $\bar{x}$. Interpreting the function f as a boolean operator, we obtain the required representation with the aid of this negation operation.

In the classification of the boolean operations we shall exclude from consideration the zero-place operations (constants 0 and 1), and also the trivial single-place operation which repeats the values of its argument. It is also natural to not differentiate between the operations which arise from the same boolean function with various permutations of its arguments. Taking account of this, we shall have a single one-place operation, negation $\bar{x}$, and eight two-place operations, multiplication xy, disjunction $x \vee y$, addition $x + y$, the equivalence operation $x \sim y$, the implication $x \supset y$, the inhibit operation $\bar{x}y$, the Sheffer operation $\bar{x} \vee \bar{y}$ and the Pierce operation $\overline{xy}$.

It is easy to verify that only two of all the listed nine boolean operations are monotone: multiplication and disjunction. Only three operations are linear; negation, addition and equivalence. Thus, we arrive at the following result.

Theorem 3. Among the nine single-place and two-place boolean operations, those of multiplication and disjunction are nonlinear (but monotone), those of negation, addition and equivalence are nonmonotone (but linear). The remaining four operations, inhibit, implication, Sheffer and Pierce, are both nonlinear and nonmonotone.

It is not difficult to derive the following important result from Theorems 2 and 3.

Theorem 4. With the use of any nonlinear operation there can be obtained either the multiplication or disjunction operation.

Let us consider the arbitrary nonlinear operation defined by the nonlinear boolean function $f(x_1, x_2, \ldots, x_n)$. The canonical polynomial of this function contains at least one product with two or more cofactors. Let us separate among all such products one of those which have the smallest length $\underline{l}$. This product contains no less than two cofactors and, consequently, has the form $x_i x_j p$, where $\underline{p}$ is the product of some set of letters (possibly empty or containing only a single letter). Keeping the letters x_i and x_j unchanged, we replace all the letters occurring in $\underline{p}$ by ones, and all the remaining letters (from the number of letters $x_1, x_2, \ldots, x_n$) by zeros. After this substitution the product which we separated out becomes $x_i x_j$ and all the remaining products of length greater than one become zero, since each of them contains at least one letter different from x_i, x_j and from the letters occurring in $\underline{p}$.

After this substitution we obtain the boolean function of two variables $\varphi(x_i, x_j)$, whose canonical polynomial has the form $x_ix_j + \alpha x_i + \beta x_j + \gamma$, where α, β, γ are the boolean constants (0 or 1). If this function is equal to x_ix_j or $x_i \vee x_j = x_ix_j + x_i + x_j$, then the theorem is proved. Otherwise, being nonlinear, this function, on the strength of Theorem 3, will inevitably be nonmonotone. But them, by Theorem 2, with the aid of the boolean operation defined by the function φ we can express the negation $\bar{x} = x + 1$. Having available the functions x_i, x_j, $x_i + 1$, $x_j + 1$ we can in the function φ replace x_i by $x_i + \beta$, and x_j by $x_j + \alpha$, after which we obtain the function $\psi(x_i, x_j)$ with the canonical polynomial $(x_i + \beta)(x_j + \alpha) + \alpha(x_i + \beta) + \beta(x_j + \alpha) + \gamma = x_ix_j + \alpha x_i + \beta x_j + \alpha\beta + \alpha x_i + \alpha\beta + \beta x_j + \alpha\beta + \gamma = x_ix_j + \delta$, where the letter δ designates the boolean constant $\alpha\beta + \gamma$. If $\delta = 0$, then $\psi(x_i, x_j) = x_ix_j$, and the theorem is proved. If, however, $\delta = 1$, then $\psi(x_ix_j) = x_ix_j + 1 = \overline{x_ix_j}$. Since we have already constructed the negation, from the last function it is again easy to obtain the product x_ix_j.

Thus, in all cases we can with the aid of the given operation construct expressions for the function xy or $x \vee y$, and consequently, also for the operations defined by them, Q.E.D. Now it is easy to derive the following condition of completeness for the sets of boolean operations.

<u>Theorem 5</u>. In order that a set of boolean operations by complete, it is necessary and sufficient that at least one nonlinear operation and at least one nonmonotone operation be included in the composition of this set.

The necessity of this condition was established above, and the sufficiency is a direct result of Theorems 2 and 4.

We agree to call the complete set of boolean operations <u>irreducible</u> if from it we cannot exclude a single operation without the set losing its property of completeness. Theorem 5 makes it easy to list all the irreducible complete sets composed from single-place and two-place boolean operations. These are four sets, each of which consists of a single operation (implication, inhibit, Sheffer operation and Pierce operation), and six complete irreducible sets consisting of two operations: combination of the operation of multiplication with each of the operations of negation, addition or equivalence, and combinations of the operation of disjunction with each of the same three operations.

The concept of completeness which we have used has made it possible in the construction of the boolean functions to use not only the arguments of these functions and the operations from the corresponding complete set, but also the boolean constants 0 and 1. If we exclude the possibility of using the constants, then there arises a new concept of completeness which we shall term <u>strong completeness</u> or <u>completeness in the strong sense</u>.

By no means all the complete sets of the boolean operations satisfy the condition of strong completeness. For example, the set consisting of the operations of addition and multiplication, being complete, nevertheless is not complete in the strong sense. It is easy to see that without the use of the constant 1 all the boolean functions constructed with the aid of this set of operations vanish at the point at which all their arguments take on zero values. Thus, with the use of only the operations of addition and multiplication (without the constant 1) there cannot be represented a whole series of boolean functions, for example the function $\bar{x}$ or the function identically equal to unity.

At the same time, the sets composed from the operations of negation and multiplication or negation and disjunction are complete not only in the conventional sense but also in the strong sense. In order to convince ourselves of this it is sufficient, obviously, to prove the possibility of representing the constants 0 and 1 with the aid of the operations indicated. This is done by the formulas $0 = \overline{x\bar{x}}$, $1 = x\bar{x}$, $1 = x \vee \bar{x}$, $0 = \overline{x \vee \bar{x}}$.

The necessary and sufficient conditions for strong completeness for the sets of boolean operations were found by Post [62]. In order to formulate these conditions, it is necessary to become acquainted with three new remarkable classes of boolean functions and the operations which they define.

The boolean function (operation) $f(x_1, x_2, \ldots, x_n)$ is termed a zero-preserving function (operation) if $f(0, 0, \ldots, 0)$ a unity-preserving function (operation) if $f(1, 1, \ldots, 1) = 1$ and a self-dual function (operation) if $f(x_1, x_2, \ldots, x_n) = \overline{f(\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)}$. The result of Post mentioned above can now be formulated as follows.

Theorem 6. In order that a set of boolean operations be complete in the strong sense it is necessary and sufficient that this set include in itself at least one nonlinear operation, at least one non-monotone operation, at least one non-zero-preserving operation, at least one non-unity-preserving operation, and at least one operation which is not self-dual.

The necessity of the conditions formulated in Theorem 6 is proved by exactly the same method as in the case of the conventional completeness: it is necessary to convince ourselves only (and this is not difficult to do) that without using the constants, with the aid of the zero-preserving operations we can construct only those boolean functions (and this means the boolean operations as well) which also preserve zero. The situation will be the same with the operations which preserve unity and with the self-dual operations. Proof of the sufficiency reduces to establishment of the possibility of construction of the constants 0 and 1 and the subsequent application of Theorem 5. The details of this proof can be found in the article of Yablonskiiy [83] (see also Glushkov [26]).

Of the nine single-place and two-place boolean operations listed above, six operations are not zero-preserving: negation, the equivalence operation, implication, and also the Sheffer and Pierce operations.

The list of operations which are not unity-preserving also includes six operations: negation, addition, the inhibit, Sheffer and Pierce operations.

Finally, all the operations other than negation are not self-dual: multiplication, disjunction, addition, implication, and also the operations of equivalence, inhibit, Sheffer and Pierce.

The Sheffer and Pierce operations possess the most remarkable property: each of them, considered individually, is a complete, in the strong sense, set of boolean operations. These sets, of course, are irreducible in the sense that from them we cannot remove a single operation without the set losing the property of strong completeness.

It is easy to show that every operation which is not zero-preserving is either also not unity-preserving or is not a self-dual operation. This implies that in any irreducible strong complete set of boolean operations there cannot be more than four (and not five, as it might seem *a priori*) different operations, and irreducible strong complete sets consisting of four different boolean operations actually do exist.

§4. APPLICATION OF BOOLEAN ALGEBRA IN THE THEORY OF COMBINATION CIRCUITS

Combination circuits are the simplest technical devices for the conversion of discrete information. Let us assume that we have at our disposal a finite number of types of signals of a particular nature (mechanical, electrical, optical, etc.) composing the so-called signal S alphabet. We shall use the term combination circuit for any device P which realizes some alphabetic operator A = A(S) in the alphabet S and satisfies the following conditions.

1. The domain of definition of the operator A is the set of words in the alphabet S having the fixed length $m \geq 1$ (depending on the choice of the device P).

2. All the input words from the domian of definition of the operator A are transformed by the circuit P into output words of the same length $n \geq 1$ (also depending on the choice of P).

3. All the letters (signals) composing the input word are applied simultaneously to the m points of the circuit P which are called its input poles, and at the same time, also simultaneously, all the letters (signals) of the corresponding output word appear at another n points of the circuit which are called its output poles. The input and output poles are numbered in a strictly fixed method and are associated with the corresponding locations of the input and output words, so that the i-th input pole ($i = 1, 2, \ldots, m$) and the j-th letter of the output word appears at the j-th output pole $j = 1, 2, \ldots, n$.

Of course, every real technical device has some internal delay, so that the condition of simultaneity of the appearance of the input and output signals in the combination system is not to be understood too literally. We are considering some abstraction of the actually encountered case in which the indicated delay can be neglected in comparison with the interval of discrete operation of the circuit, determined by the time for the replacement of one input word by another.

In practice, the combination circuits are usually characterized by the absence of memory in them. This means that the output word appears at the output poles of the circuit only for that time while the corresponding input word is applied to the input poles. After the application of the input signals has been terminated, the circuit "forgets" these signals, so they cannot affect the process of the formation of the response of the circuit to the following combination of signals applied to its input poles.

Conditions 1 and 2 impose, at first glance, very strong limitations on the alphabetic operators which can be realized by the combination circuits. In actuality, however, words of the same length (selected each time in accordance with the specific conditions) can be used to code any finite ensemble of words.

Thus, with suitable coding the combination circuits can realize any alphabetic operators with finite domains of definition.

The simplest technique for equalizing the lengths of any fixed set of words by coding consists in the suffixing (repeatedly, generally speaking) to the words of lesser length an empty word which is specially introduced into the alphabet for the purpose of bringing the number of letters composing these words up to the number of letters composing the longest word of the set in question. Of course, other techniques of resolving this problem are possible.

We note also that, with suitable treatment of the operation of the combination circuits, we can consider that the same combination circuit is capable of realizing not one, but any finite set of alphabetic operators. To accomplish this it is sufficient to separate all the input poles of the circuit into the so-called information and control poles. If we consider the transformed input word to be only that combination of input signals which is applied to the information poles, then by fixing various control words (i.e., words applied to the control poles) we will obtain different alphabetic operators which associate the output words of the circuit to the information input words.

The technique for the variation of the alphabetic operators realized by the combination circuit with the use of the control words is completely analogous to the technique described in the first chapter for the organization of the operation of the universal algorithm: to the input of the universal algorithm there is applied not only the information word to be transformed but also the control word, for which we select the representation of the specific algorithm which is to be realized.

For technical reasons it is simpler and more convenient to select the binary alphabet as the signal alphabet. In this case two types of signals are usually identified with the boolean constants 0 and 1. We shall term the combination circuits with such a signal alphabet binary, or boolean, combination circuits.

In the binary combination circuit each output signal is some boolean function of the input signals of the circuit. If the circuit has m input and n output poles, then the alphabetic operator realized by it is completely characterized by the system of n boolean functions of m variables which give the output signals on each of the n output poles as a function of the signals on the m input poles. We shall term this system of functions the output functions of the circuit in question, and the circuit itself will be termed a boolean (m, n)-terminal network.

The results of the preceding section lay the theoretical base for one of the primary problems of the theory of boolean (m, n)-terminal networks--the problem of their synthesis. The essence of the problem of the synthesis of combination circuits in general and of boolean (m, n)-terminal networks in particular amounts to the development of the methods which make it possible to construct circuits which are as complex as desired from a fixed (usually quite small) number of types of elementary combination circuits, which in the case of the binary circuits are called logic elements.

Any boolean (*m*, 1)-terminal network can be selected as a logic element. In view of what we have said above, its operation can be characterized by the output function $f(x_1, x_2, \ldots, x_m)$, which is a boolean function of *m* variables which gives the relationship of the single output signal of the element we have selected as a function of the ensemble of all its input signals. We say that the selected logic element realizes this boolean function or, correspondingly, realizes the boolean operation defined by this function.

Let us assume now that some set of logic elements has been selected. Th. synthesis of the combination circuit from the elements of the selected set amounts to the sequential connection of the output poles of some elements to the input poles of other elements in such a way that several output poles are not connected to the same input pole, and so that closed circuits are not formed along which a signal emerging from some element Q and passing, possibly, through other elements again can arrive at one of the input poles of the same element Q. Here we shall assume that we have at our disposal an unlimited number of copies of any element of the selected set so that there will be no shortage in quantity (but not number of types) of logic elements at any time.

After completing the described process of the connection of the output poles of some elements to the input poles of others, some set M of input poles and some set N of output poles are free of any connections with other poles. It is natural now to take the set M as the set of input poles and the set N as the set of output poles of the complex circuit constructed as a result of the described process.

If in the process of the connection of the poles we have observed the limitations presented above, then the circuit constructed will give an output signal on each of the _n_ poles of the set N as a completely determined boolean function of the signals on all _m_ poles of the set M. Therefore we can consider it as a combination circuit in the binary alphabet or, more precisely, as a boolean (m, n)-terminal network.

It is easy to understand that the set N of output poles of the circuit can be complemented by the poles which have been subjected to connection, which we shall term the _internal nodes_ of the circuit. With use of several types of specific physical realizations of the binary signals, we can connect several output poles to the same input pole. Ambiguity does not arise as result of the arrival of several signals at the same pole in view of the existence of the so-called _natural separation of signals_. Natural separation amounts to the fact that a zero signal is formed on a particular pole when and only when all the signals arriving simultaneously at this pole are equal to zero. If, however, even one of the arriving signals is equal to one, then the combined signal is equal to one. In this case the input signals of the circuit can, evidently, also be applied to certain of its internal nodes, as the result of which they are included in the set M of input poles of the circuit.

If the synthesized circuit has a single output pole and is thus characterized by a single output boolean function $f(x_1, x_2, \ldots, x_m)$, the described process of construction of the circuit by the method of sequential connection of the nodes in essence repeats the process of the sequential construction of the formula representing the function $f(x_1, x_2, \ldots, x_m)$ with the aid of the operations which are realized by the logic elements which we have used. The synthesis of the arbitrary boolean (m,1)-terminal network is possible if the set of indicated operations is strongly complete. Since every (m,1)-terminal network can be made up of n individual (m,1)-terminal networks, then in the case of satisfaction of the condition of strong completeness we obtain the possibility of constructing arbitrary binary combination circuits.

In practice, however, it is found as a rule that it is not difficult to apply to the synthesized circuit signals which are identically (at all instants of time) equal to zero and one using channels specially assigned for this purpose. Moreover, for the zero signal we frequently do not need any special channel, since with several physical realizations of the signals a zero signal appears on each isolated, i.e., not connected to anywhere, input pole. In this case the condition for the possibility of the synthesis of an arbitrary binary combination circuit is now not strong, but rather ordinary completeness of the set of operations which are realized by the selected logic elements. In this case, for brevity we speak of the completeness or incompleteness of the set of logic elements themselves, rather than the boolean operations realized by them.

Among the logic elements which are most frequently used in practice there are the so-called AND and OR elements which realize respectively the boolean operations of multiplication and disjunction. As a rule, along with the two-input AND and OR elements which realize the functions xy and $x \vee y$, wide use is made of the multi-input variants of these elements which realize the boolean functions $x_1 x_2 \ldots x_n$ and $x_1 \vee x_2 \vee \ldots \vee x_n$.

The boolean (1,1)-terminal network which performs the negation operation also frequently figures among the logic elements under the name of invertor. In the realization of the signals 0 and 1 in the so-called potential circuits using two different levels of electrical potential, the AND and OR circuits can be constructed with the aid of resistors and semiconductor diodes, and the inverter with the use of resistors and semiconductor triodes (transistors).

When we use two-input AND/OR/INVERT elements as the set of logic elements, the problem of the synthesis of the boolean ($\underline{m}$, 1)-terminal networks reduces to the problem of the construction of the formulas of boolean algebra which represent the output functions of these ($\underline{m}$, 1)-terminal networks. The interest is not in the construction of some circuit with the given output function (in view of what we have just said this is not difficult), but rather the construction of an adequately economical system which uses the smallest possible number of logic elements. In this case the problem of the construction of economical circuits reduces to the problem of the minimization of the formulas of boolean algebra.

Quite frequently in practice, in the construction of a particular combination circuit we have the possibility of applying to its input poles not only the input signals of interest to us $x_1, x_2, \ldots, x_n$, but also their negations $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$. In this case it is clearly sufficient for the synthesis of the circuit to have only AND and OR elements, and the problem of construction of sufficiently economical circuits is usually solved only in the class of the so-called <u>two-stage</u> circuits, i.e., circuits in which all AND elements precede the OR elements or, conversely, all OR elements precede all AND elements. Such circuits are obviously described by disjunctive or conjunctive normal forms, for which the minimization methods were discussed in §2 of the present chapter.

As an example of the synthesis of the two-stage combination circuit let us consider the synthesis of the boolean (6,1)-terminal network with the output function $f = \bar{x}yz \vee xy \vee x\overline{yz} \vee yz \vee x\bar{z}$, assuming that to the six input poles of our circuit there are applied the signals x, y, z, $\bar{x}$, $\bar{y}$, $\bar{z}$, and as logic elements we select the two-input AND and OR elements.

If we design the circuit in strict accordance with the originally given formula representing the function f, then the circuit will contain 7 AND and 4 OR elements. If, however, we minimize this formula using the Blake method as was done (precisely for this formula) at the end of §2, then it is found that the given function f can be represented by a far simpler formula: $f = yz \vee x\bar{z}$. The circuit corresponding to this formula contains in all two AND and one OR element. Representing AND and OR elements with circles having the letters C and P inside, we can represent the constructed circuit visually (Fig. 5). In the constructed circuit the input poles to which the signals $\bar{x}$ and $\bar{y}$ are applied are actually not used. Therefore the given output function can be realized by a boolean (4,1)-terminal network rather than by the (6,1)-terminal network assumed initially.

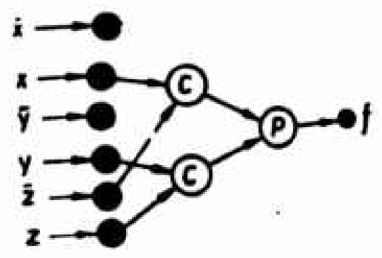

Fig. 5.

In this example the output function of the circuit to be synthesized was given in the form of some formula of boolean algebra so that the synthesis process reduced in essence only to the simplification of this formula. In practice we encounter most frequently the case when the output functions of the circuit to be synthesized are given by tables of their values. In this case the first stage of the process of circuit synthesis is the finding of some (not necessarily the most simple) formulas which represent the given functions. A universal technique for such construction is the method based on the use of the ideal disjunctive normal forms (see §2): any boolean function can be represented in the form of the disjunction of the constituents of unity co onding to those sets of values of the variables on which this .ction becomes unity. The representation obtained is then subjected to minimization.

In the case when the number of variables is relatively small, it is convenient to search for the irreducible and even the minimal disjunctive normal forms representing the given functions directly from the tables of the values of these functions. To facilitate this search use is made of special forms of writing of these tables in the form of the so-called Karnaugh maps (Veitch diagrams).

The Karnaugh map is a table with four rows designated by the various sets of values of the first two variables x and y and with 4 columns designated by the various sets of values of the last two varibles z, u. The map field (for the case of four variables) is thus divided into 16 squares which ar numbered sequentially by numbers from 0 to 15 inclusive. The Karnaugh map for four variables:

zu / xy	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

In using this map to specify a particular boolean function f(x, y, z, u), in each square there is written the value of this function (0 and 1) on that set of values of the variables whose number coincides with the number of the given square (the first two elements of the set under discussion here designate the row and the second two elements the column, at the intersection of which the square in question is located).

With this formulation in the case of the partial boolean map: designated with zero, designated with unity, and not designated at all. The last squares correspond to those sets on which the values of the function in question are not defined. In the case of the everywhere-given boolean functions, in all the squares there will be written either zero or unity, therefore these functions can be specified by the indication only of those squares in which there will be written ones, or, as we shall say, by indicating the <u>ones configuration</u> of the function in question.

The Karnaugh map is constructed so that the ones configurations which give the various elementary products are recognized very simply. For the case considered of the four variables, the ones configurations of the elementary products of length 4 (constituents of unity) reduce to separate, or as we shall say here, to <u>elementary</u> Karnaugh maps.

The corresponding configurations which give all $C_4^3 \cdot 2^3 = 32$ elementary products of length 3 are all possible pairs of elementary small squares standing in a row and thus composing a rectangle with dimensions 2×1. It is only necessary to mentally identify the opposite edges of the Karnaugh map — the upper with the lower and the left with the right. As the result of this identification it is necessary to consider, for example, that the elementary small squares with numbers 4 and 6 or with numbers 0 and 8 stand in a line, while the elementary small squares 5 and 9 or 7 and 2 must not be considered as standing in a line.

Similarly the ones configurations which give all $C_4^2 \cdot 2^2 = 24$ elementary products of length 2 are all possible combinations of four elementary small squares forming (4×1) - rectangles and (2×2) - squares, and for all $C_4^1 \cdot 2 = 8$ elementary products of length 1 the corresponding representations are given by all possible combinations of elementary small squares in (4×2) - rectangles. Here we must not forget the identification of the opposite edges of the Karnaugh map.

The elementary product corresponding to any of the ones configurations listed above is easily found, since such a product is composed of all three and only the three cofactors $(x, \bar{x}, y, \bar{y}, z, \bar{z}\ u, \bar{u})$, which become unity on all the sets of values of the variables covered by the given configuration.

Using this rule it is easy to find, for example, that to the configuration consisting of the elementary product $\bar{x}y\bar{z}$ and to the configuration ((2 x 2) -square!) consisting of the elementary squares 0, 2, 8, 10 there corresponds the elementary product $\bar{y}\bar{u}$.

When the boolean function f is given by the Karnaugh map, finding the irreducible and minimal disjunctive normal forms which represent this function reduces to finding the most economical coverings of the ones configuration which gives the function f using the ones configurations described above which correspond to the elementary products of different length (see §2).

Let us consider as an example the problem of finding such a minimal covering for the boolean function f given by the Karnaugh map

$xy \diagdown zu$	00	01	11	10
00	1	0	0	1
01	—	0	0	0
11	0	—	1	—
10	1	0	1	1

It is assumed that in the squares in which there are dashes the values of the function f can be arbitrary, so that if in the formation of a particular desired configuration it is necessary to place a one in a particular one of these squares this can always be done.

It is easy to see that all the (4 x 2) - rectangles which can be constructed on the given map include at least one zero of the function f. This means that among the elementary products of length 1 there is no implicant of the function in question. There are two (2 x 2)-squares which do not contain zeros of the function f: the "square" consisting of the four corner elements of the elementary squares and the "square" standing in the right lower corner of the map (it contains three ones and one dash). Together, these squares cover all the ones of the function f and therefore this function (with an accuracy to the indifferent values designated by the dashes) can be represented in the form of the disjunction of the corresponding elementary products $f = \bar{y}\bar{u} \vee xz$.

The disjunctive normal form found is, as it is easy to see, minimal for the function f with any possible interpretations of the indifferent values designated by the dashes.

The described technique for finding directly the minimal disjunctive forms is applicable not only for the boolean functions of four variables, but also for functions of a smaller number of variables. The Karnaugh maps of general form for functions of three and two variables:

$xy \backslash z$	0	1
00	0	1
01	2	3
11	6	7
10	4	5

$x \backslash y$	0	1
0	0	1
1	2	3

In using the first table it is necessary to mentally identify the upper and lower edges, so that the elementary squares with numbers 0; 4 and 1; 5 are to be considered neighboring.

With the aid of certain additional tricks we can construct Karnaugh maps for 5 and 6 variables. For a larger number of variables in the general case, the problem of finding the minimal representations directly from the tables of the boolean functions becomes so cumbersome that the corresponding Karnaugh maps are of little assistance. In these cases we must resort to the analytic methods for minimization of the formulas of the type of the Blake method and other similar methods.

The finding of the minimal disjunctive normal forms for the output functions of the boolean multi-terminal networks is extremely useful, not only for the synthesis of the two-stage circuits using AND and OR elements described above, but also for the synthesis of circuits using gate elements, usually termed simply gates.

Gate is the name given to the binary combination circuit with two input poles and one output pole. The operation of the gate amounts to the fact that it passes or does not pass to its output pole a signal applied at one of its input poles (termed gated pole) depending on whether there is applied to the second of the input poles (termed the control pole) a signal equal to one or, correspondingly, a signal equal to zero.

In the gate circuits (i.e., in circuits composed of gates) signals are applied to the control input poles of all input poles of all the gates which are equal to some initial variables x, y, z, ... and their negations $\bar{x}$, $\bar{y}$, $\bar{z}$, ... In addition, there is still another input pole of the circuit to which there is applied the gating input signal, identically equal to one. For the gating signals the property of natural separation (see above) is satisfied, which ensures with the application of several gating signals to the same pole taht the signal appearing on this pole will be equal to the disjunction of all these signals. The output signals of the gating circuits are also signals of the gating type.

The gating circuit for the case of a single output pole can be completely constructed using any formula which represents the output function of the circuit with the aid of the operations of multiplication and disjunction applied to the input variables and their negations (an example of such a formula might be any disjunctive normal form). With this construction, to every multiplication there corresponds a series connection, and to every disjunction there corresponds a parallel connection of gates or gate circuits composed of several gates.

If we designated a gate with a circle with the letter B inside, then a gate circuit composed in accordance with the formula $f = (x \vee \bar{y})z \vee \bar{x}y$, will have the form shown in Fig. 6. At the internal node of the circuit designated by the letter A there is generated the gating signal $x \vee \bar{y}$ (result of the parallel connection of the gates with the control signals x and $\bar{y}$). At the node B there is generated the gating signal $\bar{x}$ and at the node C the gating signal $\bar{x}y$ (result of series connection of gates with the control signals $\bar{x}$ and y). Finally, the output signal fo the entire circuit as a whole (at pole D) is the result of the parallel connection of two gate networks with the output (gated) signals $(x \vee \bar{y})z$ and $\bar{x}y$.

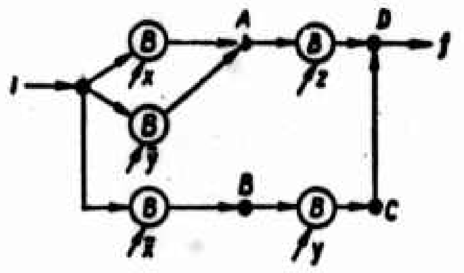

Fig. 6.

The gate circuits include the so-called relay-contact circuits which are constructed using electromagnetic relays. Gates of this type (relay contact) have two-way conductivity, transmitting the gated signals not only in the forward direction (from the gate input pole to the output pole) but also in the opposite direction. This situation gives rise to additional difficulties in the construction of the theory of the relay-contact circuits (associated with the existence of the so-called bridge circuits and the appearance of paths for signal transmission which were not initially planned). Such difficulties do not usually arise in the case of the electronic gates which do not have two-way conductivity.

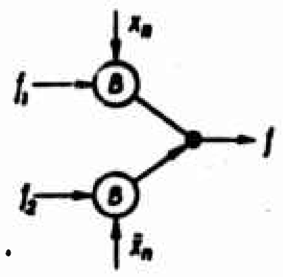

Fig. 7.

In the design of gate circuits using gates of all types the so-called cascade method (see [65]) can be of considerable assistance. This method is based on the use of the relation, valid for any boolean function f,

$$f(x_1, x_2, \ldots, x_{n-1}, x_n) = f(x_1, x_2, \ldots, x_{n-1}, 1)x_n \vee f(x_1, x_2, \ldots \ldots, x_{n-1}, 0)\bar{x}_n. \quad (34)$$

The validity of this formula is easy to see by setting $x_n = 1$ and $x_n = 0$ in it.

In application to the gate circuits, and also to the circuits constructed using the AND and OR elements, formula (34) reduces the problem of the synthesis of the circuit with the n-place out function $f(x_1, x_2, \ldots, x_n)$ to the problem of the synthesis of the circuit with two (n - 1)-place output functions $f_1(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, x_{n-1}, 1)$ and $f_2(x_1, x_2, \ldots, x_{n-1}) = f(x_1, x_2, \ldots, x_{n-1}, 0)$.

The cascade of gate circuits realizing this reduction is shown in Fig. 7. With several output functions the circuit of our (n-th) cascade becomes complicated, however the reduction process it self remains essentially the same. Continuing the reduction process, we finally construct the required gate circuit, composed in the general case (for n variables) of n cascades.

The application of a synthesis method analogous to the described cascade method permitted Shannon [80] to establish the following estimate of the number of gates (of any type) required for the realization (in the form of the output function of some gate circuit) of the arbitrary boolean function of n arguments.

Theorem 1. For any real positive number ε there exists the whole number $N = N(\varepsilon)$ such that any boolean function of $n \geq N$ varialbes can be realized in the form of the output function of a gate circuit containing no more than $(1+\varepsilon)\frac{2^{n+2}}{n}$ gates. For a similar realization with any n no more than $\frac{2^{n+3}}{n}$ gates are required.

Similar complexity estimates of circuits, but for general assumptions relative to the sets of logic elements used, were established by Lupanov (see [51], for example). It has also been shown that there exist boolean functions which cannot be realized by less than $(1-\varepsilon_n)\frac{2^{n+2}}{n}$ gates, where the quantity ε_n in this case tends to zero with unlimited increase of $\underline{n}$.

It is of interest to generalize the results presented to the case of the arbitrary boolean (m,n)-terminal networks constructed with the use of any two-input logic elements. Since every boolean (m, n)- terminal network realizes some alphabetic operator A with a finite domain of definition, the minimal possible complexity L(A) of the boolean (m, n)-terminal networks realizing the given operator A can be taken as the natural quantitative <u>complexity estimate of the operator A itself</u>. Here, in view of the absence of adequately substantiated reasons to give preference to a particular two-input logic element, it is clearly most natural in the construction of the indicated boolean (m, n)-terminal networks to make use of all the types of two-input logic elements, considering the <u>circuit complexity</u> to be the total number of logic elements composing it.

The described method is not directly suitable for the estimation of the complexity of alphabetic operators with infinite domains of definition. If, however, we are required to obtain not the absolute estimate, but only a relative practical estimate of the complexity of several alphabetic operators, we can first finitize (make finite) their domains of definition, discarding all the input words whose 1 lengths exceed some number N. This number must be selected so that the <u>probability</u> of encountering in the practical application of the operators in question input words longer than N will be sufficiently small.

If for all n = 1,2,... there are given the probabilities $\rho(n)$ of the occurrence of input words of length $\underline{n}$, then we can also proceed as follows: the given alphabetic operator A is divided into the operators A_1, A_2,... so that the operator A_n has as its domain of defination the set of all words from the domain of definition of the operator A whose length is equal to n and it acts on these words just like the operator $A(n = 1, 2, \ldots)$. Let $L(n)$ be the complexity of the operator A_n computed by the method described above. Then the complexity of the original alphabetic operator is quite naturally the infinite sum $\sum_{n=1}^{\infty} L(n)\varrho(n)$.

More rational estimates of the alphabetic operators can be obtained by using discrete automata with memory for the representation of the operators in place of the combination circuits. The fundamentals of the theory of such automata are considered in the following (third) chapter of the present book.

§5. THE CONCEPT OF PROPOSITIONAL CALCULUS

Propositional calculus is the initial and simplest portion of mathematical logic. The primary problem which mathematical logic poses to itself is the formalization of the complex thought processes which go to make up so-called logical thought. This formalization is achieved by use of the construction of logical calculus.

Every logical calculus includes in itself first of all some means for the formalization of the writing of various sorts of statements about which there is reason to say that they are true of false. It is customary in mathematical logic to call this sort of statement a proposition. The formalization, which is what we are considering here, amounts to the introduction of a rigorously defined system of symbols for the designation of various sorts of operations which make it possible to construct more complex propositions from simpler propositions. As a result of the formalization, we have the possibility of writing propositions in the form of formulas constructed from the symbols introduced by the use of definite rules.

In spite of the great importance of the formalization of the writing of the propositions, formalization in itself does not constitute the calculus. For the construction of a particular logical calculus it is necessary also to define certain formulas and operations on formulas, termed axioms and derivation rules of the corresponding calculus, which will make it possible to derive formally all possible logical corollaries from any given system of statements and will make it possible to characterize formally all the so-called identically true propositions (formulas) of the calculus in question.

In order to understand what identically true propositions are, let us consider some examples. Propositions of the type "oxygen is a gas" or "two times two is eleven" are examples of the so-called elementary constant propositions. The elementary nature of these propositions consists in the fact that they cannot be divided into simpler component parts which themselves would be propositions. Actually, the expressions "is a gas" or "two times two" are not complete propositions since the question of truth or falsity has not meaning relative to them. The term "constant" in application to the propositions presented is to emphasize that we are considering completely defined propositions relating to completely defined areas of knowledge.

We note that the truth or falsity of these propositions depends on the conditions in which they are considered and is established, as a rule, outside the limits of mathematical logic. In application to the first proposition this concept is obvious (oxygen under certain conditions can be not only a gas but also a liquid or even a solid). The second proposition, however, at first glance seems obviously false. Actually, though, all we have to do is to assume that in place of the decimal system of numbers, we are using the ternary system under the condition of retaining the names of multiplace numbers with which we are familiar, and the proposition "two times two is eleven" ($2 \times 2 = = 3 \cdot 1 + 1 = 11$) changes from false to true.

Therefore, in the applications of mathematical logic it will be necessary to specify the conditions under which a particular constant proposition is made so accurately and definitely that the truth value of this proposition cannot undergo changes in the process of obtaining various sorts of conclusions and corollaries from the proposition within the framework of the logical calculus being used. Thus, any constant proposition must be considered to be true all the time or false all the time through the entire duration of a particular logical derivation.

In propositional calculus we are not interested in the internal structure of the elementary propositions, considering them as whole units. Therefore, for their designation it is natural to make use of the individual letters of some alphabet (usually Latin). Individual letters can also be used to designate the so-called variable propositions. The term "variable proposition" in application to a particular symbol means that in place of this symbol there can always be substituted any specific constant proposition, either true or false.

Propositions, both constant and variable, can be combined into complex propositions by using as the connective the words "and", "or", "if - then", "not", etc. If variable propositions occur in the composition of the complex propositions, then with replacement of them by certain propositions the complex proposition may be true, and with replacement by others it may be false. For example, the complex proposition "A and B" where A and B are variable propositions will obviously be true in the case and only in the case when both propositions A and B are true.

However, there do exist complex propostions containing in their composition variable propositions which remain true for any values which can be given to the variable propositions mentioned. For example, the complex proposition "if it is incorrect that the proposition A is false, then the proposition A is true" remains true no matter what proposition is substituted in place of the variable proposition A. Such propositions are customarily called identically true propositions. The problem of the separation of the identically true propositions in the set of all possible propositions is a most important task of any logical calculus.

After all our preliminary remarks we turn to the construction of the propositional calculus itself.

Propositional calculus is constructed from formal objects of three types. The objects of the first type are the variable and constant propositions which are not separable into individual component parts. For their designation we shall make use of the capital Latin letters (with or without subscripts), calling them propositional letters. The objects of the second type are the propositional connectives - the formal equivalents of the connective words presented above "not", "and", "if - then". For their designation we shall make use of the corresponding symbols of negation ($\neg$), disjunction ($\vee$), conjuction ($\wedge$) and implication ($\supset$). We note that in the reading of the formulas it is more convenient to replace the implication symbol by the word "implies" and not by the words "if - then". The objects of the third type are the parentheses, which serve for expressing the order in which the propositional connective which we have listed are to operate.

Similarly to the way in which the formulas of boolean algebra were constructed in the beginning of §2, the formulas of propositional calculus are constructed from the formal objects which we have introduced. The difference lies in the use of the additional symbol $\supset$ (formal analogy of the boolean implication operation), and also in the replacement of the dot in the designation of the conjunction (multiplication) by the symbol $\wedge$, and in the use of the symbol $\neg$ standing before the negated expression in place of the bar over the negated symbol as the sign of negation.

The formulas of propositional calculus are the individusl letters and also all the expressions constructed recurrently from the already constructed formulas $\dot{A}$ and $\dot{B}$ using the rules $\neg(\mathfrak{A})$, $(\mathfrak{A}) \wedge (\mathfrak{B})$, $(\mathfrak{A}) \dot{\vee} (\mathfrak{B})$, $(\mathfrak{A}) \supset (\mathfrak{B})$. Just as in the case of the formulas of boolean algebra, in order to simplify the writing, a part of the parentheses can be dropped if this does not cause any ambiguity in the order of application of the propositional connectives. It is assumed that in the absence of parentheses is determined by the sequency $\neg$, $\wedge$, $\vee$, $\supset$ and for like parentheses the order is that of their appearance in the formula, read from left to right. In several cases an additional symbol is introduced in propositional calculus $\cong$ (or ~; read as abbreviated notation in the form $(\dot{A}) \approx (\dot{B})$ of the expression $((\mathfrak{A}) \supset (\mathfrak{B})) \wedge ((\mathfrak{B}) \supset (\mathfrak{A}))$. In using this symbol it is assumed that it occupies the last place in the sequence of symbols we have just written out (after the symbol $\supset$).

As an illustration of the method used for the reduction of the number of parentheses, we note that the formula $\neg A \vee B \wedge C \supset B \vee C$ is understood as $((\neg A) \vee (B \wedge C)) \supset (B \vee C)$, and not in any other way, the formula $A \approx B \supset C$ must be understood as $(A) \approx ((B) \supset (C))$ and not as $((A) \approx (B)) \supset (C)$, etc.

The definitions introduced above resolve only the first part of the problem of the construction of the propositional calculus - the problem of the formalization of the writing of the complex propositions. The second part of this problem - finding the method for the determination of the identically true propositions - can be resolved in two ways: the contensive and formal approaches.

In the contensive approach, which is easier to understand, we cannot for a moment forget about the contensive meaning of the concepts of the propositional letters and the propositional connectives.

In this case use is made to the maximal possible degree of the basic concept of the contensive meaning. Thus, if the propositional letter A designates a particular constant propoistion, then there is no need to remember this proposition itself, it is necessary only to know the value of the so-called *truth function* of this proposition: "true" if the proposition A is true, and "false" if the proposition A is false.

The truth function of any propositional letter denoting a variable proposition is identified with this letter itself, considering it as a *boolean variable*. Thus, the contensive meaning of the propositional letters in our construction is exhausted by their capability of taking two values: "true" and "false."

We shall associate the contensive value of the propositional connectives only with the *truth functions* of the complex propositions constructed with their use. Every formula $\underset{\bullet}{A}$ of propositional calculus can be interpreted as a formula in boolean algebra with inclusion in it of the additional operation of implication. The constant propositions appearing in the formula $\underset{\bullet}{A}$ must be replaced by the corresponding boolean constants (values of their truth functions). The symbols corresponding to the variable propositions are considered as arguments of the boolean function represented by the formula $\underset{\bullet}{A}$. This function is then termed the truth function of the complex propositions expressed by the formula $\underset{\bullet}{A}$.

On the contensive level of the construction of propositional calculus, those and only those formulas of this calculus (complex propositions) whose truth functions take the value "true" for all values of the variables are considered to be identically true.

We recall that as a result of the agreement made in §1, the value "true" corresponds to one and the value "false" corresponds to zero. Using the value tables presented in §1 for the conjuction $x \wedge y$

(table expressed by the cortege (0001)), for the disjunction $x \vee y$ (cortege (0111)), for implication $x \supset y$ (cortege (1101)), and recalling that negation transforms 1 into 0, and 0 into 1, it is easy to find the value table of the truth function for any formula of propositional calculus. This table is usually termed the <u>truth table</u> of the formula in question (or of the complex proposition defined by it). In filling in the table we use the abbreviated designations: T for true and F for false. As an example we present the truth table for the formula $A \supset B$ which defines the contensive meaning of implication (considered as a propositional connective):

A	B	$A \supset B$
Л	Л	И
Л	И	И
И	Л	Л
И	И	И

И = true;
Л = false.

From this table we see that the meaning of the term "implies" (corresponding to the propositional connective $\supset$) in propositional calculus is somewhat different than in ordinary speech. Actually, usually when we say that some proposition A implies another proposition B we have in mind that the propositions A and B are casually related with one another. Thus, the complex proposition which states that the proposition "this substance is oxygen" implies the proposition "this substance is a gas" seems to us (with the reservation made above on the gaseous nature of oxygen) both true and reasonable. At the same time, the complex propostion which states that the proposition "it is cold in the winter" implies the proposition "two times two is four" seems to us complete nonsense. However, on the strength of the truth table constructed above for the formula $A \supset B$, in propositional calculus the second of these complex propositions must be considered true to no less a degree than the first.

The reason for this presumption is not difficult to understand. Actually, in limiting ourselves by the condition of considering the

the elementary propositions only form the position of whether they are true of false, we have thereby made all the true (and all the false) elementary propositions quite indistinguishable from one another. Therefore, in particular, in the definition of the content embedded in the connective $\supset$ we are forced to operate only with the concepts of truth and falsity, and in this direction it is obviously not possible to penetrate into the inner structure of the elementary propositions of all classifications, which, or course, is necessary for the establishment of causal connections between them.

Disclosure of the internal structure of the elementary propositions and the associated increase of the capabilities for logical analysis are achieved by means of more complex logical calculus, in particular the so-called predicate calculus (see Chapter 6). As for propositional calculus, we must content ourselves with the relative poverty of its expressive capabilities, accepting this as a sort of payment for the simplicity and clarity of this calculus.

The propositional connective $\supset$ is used later on as an instrument for obtaining logical corollaries from particular formulas of propositional calculus and in the other, higher logical calculuses. Such corollaries must be true for truth of the original formulas. Therefore, the construction of the derivation must of necessity exclude the possibility (by indicating its falsity in this case) of obtaining false corollaries with truth of the original formulas. At the same time, with falsity of the original information the obtaining of any corollaries (both true and false as well) does not indicate, of course, falsity of the construction itself of the derivation. This circumstance finds its concrete expression in the truth table for the formula $A \supset B$. All we have said here will become more understandable after acquaintance with the formal aspect of propositional calculus.

The contensive aspect of propositional calculus which we have described makes it relatively easy to resolve the question on the identical truth of any complex proposition (given by more formula of the calculus): it is sufficient to sort over all possible sets of the truth values of the variable propositions composing it and verify whether on all these sets the truth function of the complex proposition in question takes the value "true." For example, the formula $A \wedge B \supset A$ will be an identically true formula of the propositional calculus on the basis of the following verification: if $A = Л$ and $B = Л$, then the formula $A \wedge B \supset A$ reduces to $Л \supset Л$, which, in view of the truth table, gives the value $И$; the same will be the case with $A = Л$ and $B = И$; with $A = И$, depending on the values of B ($Л$ or $И$), we reduce our formula either to $Л \supset И$, or to $И \supset И$, which, on the basis of the truth tables, in both cases leads to the value $И$.

We can use the technique of transformations in boolean algebra for the proof of the identical truth of the formulas of propositional calculus. We need only first replace all the implications according to the formula $(\mathfrak{A} \supset \mathfrak{B}) = (\neg \mathfrak{A} \vee \mathfrak{B})$ (see §1). With application to the example we have just considered, we obtain the following chain of transformation: $A \wedge B \supset A = \neg(A \wedge B) \vee A = (\neg A \vee \neg B) \vee A = \neg A \vee A \vee \neg B = 1 \vee \neg B = 1$ This chain proves the identical truth of the formula we started with.

The identically false propositions can be considered similarly, i.e., those (complex) propositions whose truth functions take the value "false" for all values of the variable propositions composing the given proposition. It is easy to understand that the class of all identically false propositions coincides with the negations of all possible identically true propositions.

In spite of the simplicity and the clarity, the contensive aspect of propositional calculus also has several drawbacks. First, the method of proof of the truth of the formulas, based on the sorting of all sets of values of the arguments, does not permit direct transfer of this method to the more complex calculuses in which the number of such sets may be infinite. Second, the methods which we have derived above permit the direct determination not of the identically true propositions, but of the propositions which are true with particular additional assumptions (for example, the formula $A \supset B$, which is not identically true, becomes true under the condition that the formula $(A \wedge \neg B)$. is false). But problems of this sort constantly arise in the various applications of logical calculus. We can, it is true, develop the corresponding methods within the framework of boolean algebra, however in this case still another essential difficulty is aggravated which is associated with the contensive aspect of propositional calculus — the insufficient formalization of the proof process and the very concept of the proof of the truth of particular formulas.

These deficiencies are eliminated in the completely <u>formal</u> approach to the construction of propositional calculus, which formalizes not only the method of writing the formulas (the method already described is adequate for this), but also the concept of the identically true formulas and the process of the derivation of the logical corollaries from particular propositions.

The formal aspect of propositional calculus is characterized by the fact that in this case we <u>completely avoid</u> the contensive meaning of the formulas, regarding them simply as finite sequences of individually distinguishable symbols.

For the characterization of the set of all identically true formulas we construct the <u>axiom system</u> of the calculus in question. Such systems can be chosen in various ways. We shall consider one of the most widely used axion systems of predicate calculus (see Kleene [42]).

This system includes the following axioms:

1. $A \supset (B \supset A)$.
2. $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$.
3. $A \supset (B \supset A \wedge B)$.
4. $A \wedge B \supset A$.
5. $A \wedge B \supset B$.
6. $A \supset A \vee B$.
7. $B \supset A \vee B$.
8. $(A \supset C) \supset ((B \supset C) \supset (A \vee B \supset C))$.
9. $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$.
10. $\neg\neg A \supset A$. 11. $\dfrac{\mathfrak{A},\ \mathfrak{A} \supset \mathfrak{B}}{\mathfrak{B}}$.

The first ten axioms are simply ten formulas of propositional calculus which are identically true by definition. The identical truth of the axioms presents the possibility of the substitution in place of the propositional letters A, B, C appearing in them of any forulas of propositional calculus (not necessarily true). Such a substitution, by definition, will not destroy the identical truth of the formula (axiom) subjected to this substitution.

The eleventh axiom has its own specific nature. This is the so-called rule of derivation which makes it possible, by definition, to consider the truth of formula $\dot{B}$ proved if the truth of formulas $\dot{A}$ and $\dot{A} \supset \dot{B}$ has already been proved previously. If the formulas $\dot{A}$ and $\dot{A} \supset \dot{B}$ are in this case identically true then formula $\dot{B}$ will also be identically true. It is presumed by definition that all identically true formulas (and only such formulas) of propositional calculus can be obtained from the axioms as the result of the described substitutions and applications (multiple, generally speaking) of the derivation rule 11.

It in no wise follows a priority that the set formally characterized in this fashion of all identically true formulas of propositional calculus will coincide with the set of all identically true formulas defined contensively above. Since the formal identical truth of the formulas is established by the procedure of the derivation or proof, they are also termed (formally) *provable* formulas or (formal) theorems.

The formulas which are identically true in the contensive sense we shall for brevity term simply *contensively true*, contrasting them with the *formally true* (i.e., formally provable) formulas.

The concept of formal derivation (proof) can be extended to the case when, in addition to the axioms, there is given also some quantity of formulas $\underset{\cdot}{A}_1$, $\underset{\cdot}{A}_2$ ..., $\underset{\cdot}{A}_n$ of the propositional calculus as *conditionally true formulas*. These formulas are not derivable from the axioms (not formally provable) and therefore are not formally true formulas. The presumption on their truth is of a conditional nature and is retained only in the course of the derivation in question. In contrast with the axioms of propositional calculus, in these formulas we cannot, generally speaking, replace the propositional letters appearing in them by arbitrary formulas. In other words, conditional truth, in contrast with formal truth, does not have an *identical* nature.

However, the rules of the derivation themselves are in essence retained as before. The primary role, as before, is played by the concept of the *direct corollary* (axiom 11): formula $\underset{\cdot}{B}$ is termed the direct corollary of the formulas $\underset{\cdot}{A}$ and $\underset{\cdot}{A} \supset \underset{\cdot}{B}$. We say that the formula $\underset{\cdot}{C}$ of propositional calculus is derived from formulas $\underset{\cdot}{A}_1$, $\underset{\cdot}{A}_2$, ..., $\underset{\cdot}{A}_n$, if it can be obtained from these formulas and axioms 1-10 of propositional calculus as the result of the application (a finite number of times) of the rule of the direct corollary. More precisely, in the case being considered those formulas and only those formulas will be derivable which are obtained as the result of the sequential application of three rules. 1. Any of the formulas $\underset{\cdot}{A}_1$, $\underset{\cdot}{A}_2$, ..., $\underset{\cdot}{A}_n$ is derivable. 2. Any of the axioms 1-10 (with account for the possibility of the substitution of any formulas in place of the letters appearing in them) is derivable. 3. If the formulas $\underset{\cdot}{A}$ and $\underset{\cdot}{A} \supset \underset{\cdot}{B}$ are derivable, then formula $\underset{\cdot}{B}$ will also be derivable. The chain of formulas obtained as the result of the sequential application of these three rules, which terminates with some formula $\underset{\cdot}{C}$, is termed the formal derivation of this formula.

For the designation of the derviability we make use of the special symbol $\vdash$ (read as "gives"), to the left of which there are written the conditionally true formulas and to the right are written their corollaries: $\underset{\cdot}{A}_1, \underset{\cdot}{A}_2, \ldots, \underset{\cdot}{A}_n \vdash \underset{\cdot}{C}$. The axioms of propositional calculus are not written out explicitly here (the possibility of their use in the derivation is really included in the symbol $\vdash$) so that for any formally true formula $\underset{\cdot}{B}$ we can write $\vdash \underset{\cdot}{B}$. In other words, the formally true formulas are considered derivable from the empty set of (conditionally true) fromulas. Therefore axioms 1-10 can also be considered as sort of rules of derivation which derive the formulas representing them from the empty set of formulas.

We shall present very simple examples of the formal derivation, numbering the sequential steps.

1. $A \supset (A \supset A)$ (axiom 1, in which the letter B is replaced by the letter A).

2. $(A \supset (A \supset A)) \supset ((A \supset ((A \supset A) \supset A)) \supset (A \supset A))$ (axiom 2, which the letter B is replaced by the formula $A \supset A$, and the letter C is replaced by the letter A).

3. $(A \supset ((A \supset A) \supset A)) \supset (A \supset A)$ (application of the derivation rule 11 to the formulas obtained in steps 1 and 2).

4. $A \supset ((A \supset A) \supset A$ (axiom 1, in which the letter B is replaced

by the formula $(A \supset A)$).

5. $A \supset A$ (application of the derivation rule 11 to the formulas obtained in the preceding two steps).

This chain of formulas is, on the basis of the definition, the formal proof of the formula $A \supset A$, i.e., its derivation from the empty set of (conditionally true) formulas. Thus, the formula $A \supset A$ belongs to the number of the formally true formulas ans it can be written as $\vdash A \supset A$.

Another example is the derivation of the corollaries from the three conditionally true formulas A, B, $A \supset (B \supset C)$. The formula C can be derived from these formulas after 5 steps.

1. A (first given (conditionally true) formula).

2. B (second given formula).

3. $A \supset (B \supset C)$ (third given formula).

4. $B \supset C$ (direct corollary (from derivation rule 11) from formulas 1 and 2).

5. C (direct corollary of formulas 2 and 4). Thus, formula C is derivable from formulas A, B, $A \supset (B \supset C)$, and we can write $A, B, A \supset \supset (B \supset C) \vdash C$.

Although the conditionally true formulas do not possess identical truth, still, as it is easy to see, in the final writing of the (conditional) derivability with the use of the symbol $\vdash$ any letter can be replaced by an arbitrary formula of propositional calculus, if such a replacement is performed simultaneously both to the left and to the right of the derivability symbol. Replacement in only one side will lead, generally speaking, to error.

In similar fashion we can prove the relations

chain conclusion $A \supset B, B \supset C \vdash A \supset C$; (35)

permutation of premises $A \supset (B \supset C) \vdash B \supset (A \supset C)$; (36)

importation $A \supset (B \supset C) \vdash A \wedge B \supset C$; (37)

exportation $A \wedge B \supset C \vdash A \supset (B \supset C)$; (38)

contraposition $A \supset B \vdash \neg B \supset \neg A$; (39)

$\wedge$ -insertion $A, B \vdash A \wedge B$; (40)

weak $\neg$ -removal $A, \neg A \vdash B$. (41)

If we designate by Γ an arbitrary finite ensemble of formulas of propositional calculus, then, using somewhat more complex method of proof (induction during the derivation) we can obtain the following result (the so-called deduction theorm).

Theorem 1. If in the propositional calculus formula B is derivable from the combination of formulas Γ and A then the formula A $\supset$ B is derivable from Γ.

Two universal proof schemes are also of importance in the theory of proofs.

1. Proof by means of analysis of cases: if Γ, A $\vdash$ C and Γ, B $\vdash$ C, then

$$\Gamma, A \vee B \vdash C. \quad (42)$$

2. Reduction and absurdum: if Γ, A $\vdash$ B and Γ, A $\vdash$ $\neg$ B, then

$$\Gamma \vdash \neg A. \quad (43)$$

It is easy to verify that all the axioms 1-10 of the propositional calculus are contensively true formulas. In other words, the truth functions corresponding to them take the value "true" for all values of the variables. This property is obviously retained with substitutions of any formulas of the propositional calculus in place of the letters appearing in the axioms. From the truth table for implication it follows directly that from the contensive truth of the formulas $\dot{A}$ and $\dot{A} \supset \dot{B}$ there follows the contensive truth of the formula $\dot{B}$. But then, obviously, all the provable (formally true) formulas will inevitably be contensively true. The reverse is also true (although much more complex to prove), so that the following important result can be formulated.

Theorem 2. In the formal construction of propositional calculus using the system of axioms 1-11, all those and only those formulas of this calculus will be provable (formally true) which are identically true in the contensive sense.

Theorem 2 contains, actually, two results relative to the selected system S of axioms of the propositional calculus. The first result is that the system S is contensively consistent or, other words, with the aid of the system S we cannot prove a single formula which is not a contensively true formula.

The second result states the contensive completeness of the system of axioms S: there is no single contensively true formula of propositions which cannot be proved formally with the aid of this system of axioms.

The question arises of whether it is possible to determine the properties of consistency and completeness purely formally without resorting to the contensive constructions. It is found that it is possible.

It is natural to term the system of axioms of propositional calculus formally consistent if with its aid we cannot derive any formula $\underset{\cdot}{A}$ together with its negation $\neg \underset{\cdot}{A}$, and formally inconsistent in the opposite case.

From the property of weak $\neg$-removal it follows directly that in the case of the formal inconsistency of the system of axioms any formula of propositional calculus would be formally provable. Since, as the result of Theorem 2, for the system S the letter situation does not occur, then this system is not only contensively consistent but also is formally consistent, or, as is often said, is simply a consistent system of azioms.

The property of formal completeness for the system of axioms can be defined as follows: a system of axiom is termed formally complete (or complete in the restricted sense) if the addition to this system as a new axiom of any formula which is not provable in the system leads to the system of axioms thus expanded being formally inconsistent. In this case it is usually presumed that the original axiom system was formally consistent.

It can be shown that the system of axioms 1-11 of propositional calculus which we have introduced is not only a contensively but also a formally complete system of axioms. Under the condition of satisfaction of the property of contensive consistency and with the use of only rule 11 as a derivation rule, from the property of formal completeness, since otherwise any nonprovable contensively true formula could be used for consistent expansion of the original axiom system.

In the axiom system we have chosen there is not a single redundant axiom. More precisely, no one of the formulas 1-10 can be formally proved with the aid of the ensemble of all the remaining axioms. This property is termed the property of independence of the axioms of the selected system. The property of independence is proved separately for each axiom with the aid of the construction of a contensive interpretation for which this axiom is not utilized while all the remaining axioms are utilized.

We note, finally, that although the joining of unprovable formulas as new axioms to the propositional calculus axiom system S which we have chosen, on the strength of the property of formal completeness of this system, destroys the property of its formal consistency, nothing prevents us from joining to the system S the unprovable (in S) formulas $\dot{A}_1, \ldots, \dot{A}_m$ as conditionally true formulas rather than as identically true formulas. It can be shown that inconsistency (the possibility of deriving some formula together with its negation) in this case arises when and only when the conjunction $\mathfrak{A}_1 \wedge \mathfrak{A}_2 \wedge \ldots \wedge \mathfrak{A}_m$ is an *identically false formula*.

As the result of this joining, there arises a formal theory which goes beyond the framework of mathematical logic proper, since the joined formulas $\underset{\cdot}{A}_1$, $\underset{\cdot}{A}_2$, ..., $\underset{\cdot}{A}_m$ are not true in the strictly logical sense. If in our constructions there is some particular contensive meaning, then the contensive truth of the formulas $\underset{\cdot}{A}_1$, $\underset{\cdot}{A}_2$, ..., $\underset{\cdot}{A}_m$ must be postulated or have some clearly extra-logical basis. In that case it is natural to consider these formulas as *axioms* of the formal theory constructed on their basis. In order not to confuse them with the logic axioms 1-11 themselves, the latter are in this case termed not axioms, but axiom *schemes*, thereby emphasizing that each of the axioms 1-11 is actually a whole *set* of axioms obtained from the formula corresponding to this axiom as the result of the replacement by arbitrary formulas of the propositional calculus of the letters appearing in this formula.

Chapter 3

THEORY OF AUTOMATA

§1. ABSTRACT AUTOMATA AND AUTOMATON REPRESENTATIONS

Let us consider the alphabetic transformations realizable by discrete information processors which put out some output signal (letter of the output alphabet) in response to each input signal (letter of the input alphabet). Such processors, considered without regard to their internal structure, are customarily termed abstract automata.

For the specification of an abstract automaton, three sets must be given: the input alphabet $\underset{\cdot}{X}$, the output alphabet $\underset{\cdot}{Y}$ and the set of internal states of the automaton, which we shall denote by the letter $\underset{\cdot}{A}$. The automaton operates in discrete time, whose sequential moments are conveniently identified with the sequential natural numbers $t = 0, 1, 2, \ldots$ (which we can always do by suitable choice of the time measurement unit).

At every given instant of discrete automaton time $t = 0, 1, \ldots$ the automaton A is in some definite state $a = a(t)$ of the set $\underset{\cdot}{A}$ of its internal states, which for brevity we shall term the state set of the automaton A. The state $a_0 = a(0)$ at the initial instant of time $t = 0$ is termed the initial state of the automaton A. If the initial state remains unchanged during any experiments with the automaton, then this automaton is termed an initial automaton. Since, however, in practice we do not consider any automata other than initial, the term "initial" is frequently dropped.

At every instant t of automaton time, beginning with t = 1, to the input of the automaton there is applied as the input signal one of the letters of the input alphabet $\dot{X}$ x = x(t). The finite ordered sequences of the input signals x(1)x(2) ... x(k) of the automaton are termed the input words of this automaton. Any input word from some a priori fixed set of admissible input words can be applied to the input of the automaton.

Any admissible word p = x(1) x (2) ... x(k), applied to the input of a given initial automaton A causes the appearance at the output of the automaton of the output word q = y(1)y(2) ... y(k), which is some ordered finite sequence of the output signals of the automaton A (letters of its output alphabet $\dot{Y}$) having the same length as its corresponding input word p and which is uniquely determined by the input word p. The resulting correspondence φ between the admissible input words p and their corresponding output q is termed the (alphabetic) representation induced by the initial automaton A in question.

This representation φ is uniquely determined by specifying the two functions δ and λ, termed respectively the switching function and the output function of the automaton A in question.

The switching function determines the state a(t) of the automaton at any instant of discrete automaton time t from the input signal x(t) at that same instant and from the state a(t − 1) at the preceding instant of automaton time

$$a(t) = \delta(a(t-1),\ x(t)). \qquad (44)$$

The output function determines the variation of the output signal y(t) of the automaton with these same variables

$$y(t) = \lambda(a(t-1),\ x(t)). \qquad (45)$$

Specifying any input word p = x(1) x (2) ... x(k) and initial state a(0) of the automaton, with the aid of relations (44) and (45) we can sequentially determine all the letters of the corresponding output word

$$q = \varphi(p) = y(1)y(2)\ldots y(k).$$

Thus, the relations (44) and (45) actually define the representation φ induced by the automaton.

The switching and output functions are usually the abstract partial functions $\delta(a, x)$ and $\lambda(a, x)$ which specify the single-valued representations of some set of pairs (a, x) $(a \in \underset{\cdot}{A}, x \in \underset{\cdot}{X})$ in the sets $\underset{\cdot}{A}$ and $\underset{\cdot}{Y}$ respectively. Admissible input words are those and only those input words p on which with the aid of the function δ and λ using the method described above there are determined their corresponding output words $\varphi(p)$.

The automaton is termed finite if all three of the sets $\underset{\cdot}{A}$, $\underset{\cdot}{X}$, $\underset{\cdot}{Y}$ defining it are finite. Since we limit ourselves almost exclusively to the consideration of finite automata, the word "finite" is often dropped. The automaton is called completely determinate if its switching and output functions are given on all pairs (a, x), and partially determinate otherwise.

The finite automata are customarily specified by two tables, termed respectively the switching table and the output table of the automaton. The rows of both tables are designated by the different letters of the input alphabet $\underset{\cdot}{X}$ of the automaton, and the columns by the different states of the automaton. At the intersection of the x-th row and the a-th column of the switching table there stands the element $\delta(a, x)$, i.e., some state of the automaton from the set of its internal states, and at the intersection of the x-th row and the a-th column of the output table there stands the element $\lambda(a, x)$, i.e., some letter of the output alphabet $\underset{\cdot}{Y}$ of the automaton. Thus the specification of the switching and output tables determines both the sets

$\dot{X}$, $\dot{Y}$, $\dot{A}$, and the switching and output functions of the automaton. For fixing the initial state it is usually customary to designate the first column on the left of both these tables with this state. Thus, the use of the two tables makes it possible to specify any finite automata, including the initial automata.

Another method of specifying the finite automata which provides better visualization is that of the directed graphs. The vertices of the graph (shown as circles on the figures) are identified with the various states of the automaton. The arrow connecting the vertex $\underline{i}$ with the vertex $\underline{j}$ signifies that there exists an input signal $\underline{x}$ which transfers the automaton from the state $\underline{i}$ into the state $\underline{j}$, i.e. satisfying the relation

$$j = \delta(i, x).$$

In order to differentiate precisely which input signals cause the transfer of the automaton from state $\underline{i}$ into the state $\underline{j}$, the arrow connecting the graph vertices corresponding to these states are flagged with the symblos of these input signals. The output signal $\underline{y}$ determined by the pair (i, x) is usually placed on the graph alongside the input signal $\underline{x}$ and to differentiate it from the input signals it is inclosed in parentheses.

Let us consider an example of the specification of a finite automaton using the switching and output tables of the directed graph. Let us choose for this purpose the relatively simple automaton with three internal states 1, 2, 3, two input signals x, y and two output signals u, v. We assume that this automaton is specified by the switching and output tables

	1	2	3
x	2	3	3
y	3	2	2

;

	1	2	3
x	u	u	v
y	v	u	u

.

The directed graph shown in Fig. 8 corresponds to these tables.

The automata we have considered above are customarily termed Mealy automata (from the name of the scientist who first considered several questions associated with the functioning of such automata; see [55]). In practice we frequently have to deal also with somewhat differently defined automata which are termed Moore automata (see [57]).

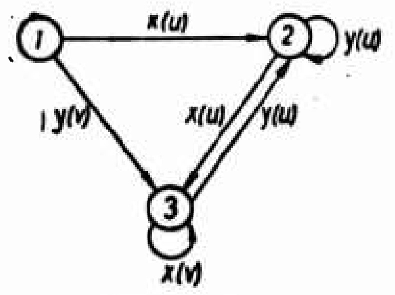

Fig. 8

The Moore automata differ from the Mealy automata only in the method of defining their output functions. In place of the relation

$$y(t) = \lambda(a(t-1), x(t)),$$

which defines the output signal for the Mealy automata, in the case of the Moore automata we use a somewhat different relation

$$y(t) = \mu(a(t)). \qquad (46)$$

Which the aid of relation (46) and the previously written relation (44), just as in the preceding case, there is determined the representation induced by any given Moore automaton.

For reasons which will be considered later, we call the function $y = \mu(a)$ the shifted output function of the Moore automaton. The value of this function for any state a is customarily termed the label of this state. The finite Moore automata are conveniently specified with the use of the so-called labelled switching tables. The labelled switching table is nothing other than the conventional switching table of an automaton in which above the symbols of the states designating the various columns of the table there are placed the labels of these states. For example, the labelled switching table

$$\begin{array}{c|ccc} & u & u & v \\ & 1 & 2 & 3 \\ \hline x & 2 & 3 & 3 \\ y & 3 & 2 & 2 \end{array}$$

specifies the Moore automaton having the same switching table as the Mealy automaton in which the output signal u corresponds to states 1 and 2, and the output signal v corresponds to the state 3. In the representation of the Moore automata with the use of graphs, the symbols of the output signals label the corresponding vertices of the graph, and not the lines as in the case of the Mealy automata.

We agree to consider that the delivery of the output signals in the Moore automaton begins at the instant of time $t = 1$ (at not at the instant of time $t = 0$). With this condition, for any Moore automaton A_1 it is not difficult to construct that Mealy automaton A_2 having the same switching table and inducing the same representation as the automaton A_1.

Actually, if $\delta(a, x)$ is the switching function and $\mu(a)$ is the shifted output function of the Moore automaton A_1, then we can define the Mealy automaton A_2 by specifying its switching function $\delta(a, x)$ and output function $\lambda(a, x) = \mu(\delta\ (a, x))$. Then

$$y(t) = \lambda(a(t-1),\ x(t)) = \mu(\delta(a(t-1),\ x(t))) = \mu(a(t)),$$

which proves that the automata A_1 and A_2 react completely identically to any sequence of input signals. The construction described is termed the interpretation of the given Moore automaton <u>as a Mealy automaton</u>. The physical meaning of such an interpretation (in real automata) consists in the shift of the automaton time by one elementary interval of time, on the strength of which in the constructed Mealy automaton A_2 the output signals lead by one unit of automaton time their corresponding output signals in the Moore automaton A_1. It is precisely for this reason that the output functions of the Moore automaton are termed the shifted output functions.

The described time shift maked it possible to consider the Moore automata as a particular case of the Mealy automata every time that we are interested not in the real time of the appearance of a particular output signal, but only in the sequence of succession of the output signals in time. It is exactly this situation which we encounter in the abstract theory of automata, being interested only in the representations induced by the automata and the switchings in their memory, and not in the method of composition of a given automaton from the elementary automata available to us.

In the resolution of the latter question, constituting the subject of the so-called structural theory of automata, the Mealy automata are to be considered as a separate class of automata which is not an intrinsic subclass of the class of all Mealy automata. The difference between these two classes of automata in structural theory is due to the fact that in the Mealy automata the output signal arises simultaneously with the input signal which induces it, while in the Moore automata there is a delay of one unit of automaton time.

The possibility of the interpretation of every Moore automaton as a Mealy automaton in the abstract theory of automata does not indicate, or course, the existence of the reverse possibility. Nevertheless, for any Mealy automaton A we can construct a Moore automaton B which will induce the same representation as the automaton A. Here, in contrast with the preceding case, the set of states of automaton B will not, generally speaking, coincide with the set of states of automaton A, althought it will be finite whenever the latter set is finite.

Actually, let us assume that there is given the arbitrary Mealy automaton A with the set of states A, the input alphabet X, the output alphabet Y, the switching function $\delta(a, x)$, the output function $\lambda(\alpha, x)$ and the initial state a_0. We agree for simplicity of notation that here and hereafter we shall use in place of the switching function the multiplication symbol, designating the value of the function $\delta(a, x)$ by the product ax.

Let us construct the Moore automaton B, selecting as the set B, of its states the set consisting of the initial state a_0 and the set of all possible pairs (a, x) where $a \in A$ $x \in X$. The input and output alphabets of the automaton B coincide respectively with the input and output alphabets of the automaton A. We determine the switching function of the automaton B, setting

$$a_0x = (a_0, x) \text{ and } (a, x_i)x_j = (ax_i, x_j).$$

We determine the shifted output function $\mu(b)$ of the automaton B on each state $b = (a, x)$ which differs from the initial state a_0 with the aid of the relation $\mu(b) = \lambda(a, x)$. In the initial state the value of the function μ can be selected arbitrarily. As a result there is constructed some Moore automaton B.

It is not difficult to see that theautomaton B induces the same representation as the automaton A. Actually, let us designate by the letter φ the representation induced by the automaton A and by the letter ψ the representation induced by the automaton B. Assume that for any input word $p = p_1x_i$ of length $n \geq 1$ it has already been proved that $\varphi(p) = \psi(p) = q$ (for the input word of length 1, i.e., for any single-letter word $\underline{x}$, obviously, $\varphi(x) = \psi(x) = \lambda(a_0, x)$).

Let us consider the reaction of both automata to the arbitrary word px_j or length $n + 1$. Let us agree here and hereafter to designate with the word $\underline{a\ell}$ the state into which there transfers an automaton which was initially in the state $\underline{a}$ if to its input there is applied sequentially, letter after letter, the arbitrary word ℓ. As a result of the definition of the switching function in automaton B, $\alpha_0 p =$ $= (\alpha_0 p_1, x_i)$. After application to the automata A and B of the input signal x_j the automaton A delivers the output signal $y = \lambda(a_0 p, x_j)$. Automaton B will obviously transfer into the state

$$b = (a_0 p_1 x_i, x_j) = (a_0 p, x_j)$$

and will deliver the output signal $\mu(b)$, equal, as a result of the definition of the function μ, to the signal $\lambda(\alpha_0 p, x_j)$.

Thereby it is shown that the automata A and B react identically to any input word to length n + 1. Performing an induction with respect to n, we come to the conclusion that the representations induced by the automata A and B are identical. This conclusion is valid not only for the conventional (completely determinate) automata, but also for the partial automata.

Let us characterize in more detail the representations induced by the automata. We note that the requirement for the arrival of an input signal and the departure of an output signal at every instant of automaton time, which at first glance is not satisfied in any specific automata, in actuality is easily satisfied if we introduce special letters for the designation of empty input and output signals (i.e., the absence of any real physical signals) and consider these letters on a par with the other letters of the input and output alphabets.

It is easy to see that the representation φ induced by the arbitrary Moore or Mealy automaton satisfies two conditions:

1) to any word ℓ in the input alphabet $\dot{X}$ the representation φ associates a word $\varphi(\ell)$ in the output alphabet $\dot{Y}$ which has a length identical to that of the word ℓ;

2) if the word ℓ_1 coincides with the initial segment of the word ℓ, then the word $\varphi(\ell_1)$ is the initial segment of the word $\varphi(\ell)$.

Let us term the conditions just formulated the automaticity conditions of the representation φ and every correspondence between the words in the alphabets $\dot{X}$ and $\dot{Y}$ which satisfy these conditions an automaton representation or automaton operator.

It is not difficult to show that every automaton representation can be induced with the aid of some abstract automaton (not necessarily finite).

Let the automaton correspondence φ map the set of words in the alphabet $\dot{X} = (x_1, x_2, \ldots, x_n)$ into a set of words in the alphabet $\dot{Y} = (y_1, y_2, \ldots, y_m)$. Let us construct the automaton A whose internal states will be all possible words in alphabet $\dot{X}$ and the initial state will be the empty word e (word of zero length, consisting of an empty set of letters). The switching function δ is determined trivially: if ℓ is any state of the automaton (word in the alphabet $\dot{X}$), and x_i is any input signal, then the value of the function $\delta(\ell, x_i)$ is assumed equal to the word ℓx_i. After determining the output function λ by the relation $\lambda(\ell, x_i) = y_j$, where y_j is the last letter of the word $\varphi(\ell x_i)$, we obtain an automaton which realizes the original mapping φ.

If the mapping φ of the set of words in the alphabet $\dot{X}$ into the set of words in the alphabet $\dot{Y}$ is given by a partial automaton, then it will be, of course, only a partial mapping, not determinate on all the words. However, as before, both conditions of automaticity will be satisfied for this mapping under the additional assumption that $\varphi(\ell)$ exists. In this case the second condition of automaticity takes a stronger form: if $\varphi(\ell)$ exists and ℓ_1 is the initial segment of the word ℓ, then $\varphi(\ell_1)$ exists and coincides with some initial segment of the word $\varphi(\ell)$.

We shall term the rephrased conditions the automaticity conditions of the partial mapping φ, and every partial mapping satisfying these conditions will be termed a <u>partial automaton mapping</u>.

It is easy to setablish the validity of the following proposition.

<u>Theorem 1</u>. Every partial automaton mapping can be induced with the aid of some partial automaton (not necessarily finite).

This proposition is proved by exactly the same method as in the case of the complete mapping. The difference is that the states of the partial automaton are considered to be not all the words of the input alphabet, but only those on which the mapping φ is determinate.

At first glance the automaticity conditions severely narrow the class of mappings which can be specified with the aid of the abstract automata. It is well known, in particular, that the requirement for equality of the lengths of the input and output words is not satisfied for a large portion of the algorithms which must be satisfied by particular specific automata. This difficulty, seeming very serious at first glance, in actuality is easily removed with the aid of recoding of the input and output information on the basis of a very simple technique.

The standard technque for the conversion of any partial correspondence φ between words in the alphabets $\underset{\cdot}{X}$ and $\underset{\cdot}{Y}$ into a partial automaton correspondence is based on the introduction into the alphabets $\underset{\cdot}{X}$ and $\underset{\cdot}{Y}$ of the letter α which was not contained in them previiously. The letter α is termed an <u>empty word</u>. The appearance of the empty word at the automaton input corresponds to the case when in actuality nothing is applied to the automaton input. Similarly the appearance of the empty word as an output signal signifies the absence of any signal at the automaton output.

Let us consider the arbitrary word ℓ of length $\underline{n}$ in the alphabet $\underset{\cdot}{X}$, to which the initially specified partial mapping φ associated the word $q = \varphi(\ell)$ of length $\underline{m}$ in the alphabet $\dot{Y}$. Let us designate by the letter ℓ_1 the word in the alphabet $\dot{X}_1 = \dot{X} \cup (\alpha)$, which is obtained as a result of the suffixing to the word ℓ on the right $\underline{m}$ exemplars of the letter α. Similarly, we use the word q_1 to designate the word in the alphabet $\dot{Y}_1$ $\dot{Y} \cup (\alpha)$, obtained as a result of the prefixing to the word $\underline{q}$ on the left $\underline{n}$ exemplars of the latter α. We term this technique the <u>standard technique for equalizing word lengths</u>.

Let us determine a new partial mapping φ_1 between words in the alphabets $\dot{X}_1$ and $\dot{Y}_1$, setting $q_1 = \varphi_1(\ell_1)$ and repeating this technique for any word ℓ in the alphabet $\dot{X}$ on which the mapping φ is determinate. We further define this correspondence on all the initial segments $\ell_1^{(i)}$ of the word ℓ_1, assuming that $\varphi_1(\ell_1^{(i)}$ coincides with the initial segment of the word $\varphi_1(\ell_1)$ having a length equal to $\ell_1^{(i)}$.

With this redefinition there arises the danger of loss of uniqueness of the mapping φ, since the word $\ell_1^{(i)}$ can occur not only as the initial segment in the original word ℓ_1, but also as the initial segment in another word, for example in the word s_1 obtained as the result of the application of the standard technique of equalizing word lengths from some word $\underline{s}$ in the alphabet $\dot{X}$.

Since the word s_1 has the form $s_1 = s\alpha\alpha \ldots \alpha$, and the word ℓ_1 has the form $\ell_1 = \ell\alpha\alpha \ldots \alpha$, where the words $\underline{s}$ and ℓ do not contain the letter α, then $p = s = \ell$ if the word $\ell_1^{(i)}$ has on the right at least one letter α : $\ell_1^{(i)} = p\alpha \ldots$. In this case, consequently, the words s_1 and ℓ_1 must coincide with one another and there is no danger of ambiguity arising.

It remains, thus, to consider the case when the word $\ell_1^{(i)} = p$ consists exclusively of letters of the alphabet $\dot{X}$. In this case the length of the word $\underline{p}$ will clearly not exceed the lengths of the words ℓ and $\underline{s}$. But then, as a result of the standard technique for the equalizing of word lengths, the initial segments of the words $\varphi(\ell)$ and $\varphi_1(S_1)$, having a length equal to that of the word $\ell_1^{(1)} = p$, consist entirely of the letters α and, consequently, coincide with one another. Thus, the occurrence of ambiguity is excluded again in this case.

The partial mapping φ_1 between words in the alphabets X_1 and Y_1 which we have constructed satisfies both conditions of automaticity for partial mappings on the basis of the method of construction itself and is, consequently, the sought partial automaton mapping.

The described technique for the transformation of any partial mapping into an automaton mapping is universal, however, precisely because of its universality it does not always lead to the most economical (from the point of view of the use of additional letters) solution. This circumstance is particularly easily clarified for the case when the original partial mapping φ itself satisfied both conditions of automaticity. It is clear that the most economical solution in this case will be $\varphi_1 = \varphi$. However, the described standard method (which we use, of course, in this case as well) leads to an unnecessary increase of the lengths of the original words which participate in the correspondence.

Thus, the universal technique found does not avoid the necessity for looking for more economical solutions. Such economic solutions are usually found by adding empty letters to the words gradually, step by step, rather than all at once in the quantity provided for by the standard technique for equalizing word lengths, checking at each step for satisfaction of the automaticity conditions and stopping as soon as they are satisfied for the first time. Such an <u>improved</u> technique for equalizing word lengths will lead sooner or later to the appearance of the automaton mapping.

Of considerable interest is the problem of finding the economical recoding of the mapping, given on a particularalgorithmic language (for example, on the language of the normal algorithms) for the purpose of converting it into sn automaton correspondence, and also the problem of the construction of the theory of algorithms which satisfy the conditions of automaticity and therefore are termed automaton algorithms for short. One of the possible approaches to the theroy of automaton algorithms id developed in the following section.

§2. EVENTS AND REPRESENTATION OF EVENTS IN AUTOMATA

Let A be an arbitrary (partial, generally speaking) initial automaton, φ the mapping induced by it. For each letter y_i of the output alphabet $\underset{\cdot}{Y} = (y_1, y_2, \ldots, y_m)$ of automaton A let us consider the set R_i of all words ℓ in the input alphabet $\underset{\cdot}{X} = (x_1, x_2, \ldots, x_n)$ of this automaton for which the word $\varphi(\ell)$ is defined and ends with the letter y_i.

Let us term the set R_i thus defined an event, represented in the (partial) automaton A by the output signal y_i $(i = 1, 2, \ldots, m)$. If M is any set of output signals, then we shall term the union of events represented by all elements of this set an event, represented in the partial automaton A by the set M.

It is easy to see that the sets R_i are disjoint and that the set S of all words in the alphabet $\underset{\cdot}{X}$ which do not occur in even one of the sets R_i $(i = 1, 2, \ldots, m)$ consists of all words forbidden for the given partial automaton. Here and herafter we use the term forbidden for all words in the input alphabet which when applied to the input of the given partial automaton lead for at least one component of their input signal to an output signal which is not defined in the automaton. We agree to call the ensemble of all forbidden words S the forbidden domain of the given partial automaton A. We agree also to term any set of words in the alphabet $\underset{\cdot}{X}$ an event in this alphabet.

From the definitions introduced, we can formulate the result obtained above in the form of the following proposition.

Theorem 1. Specification of the partial automaton mapping φ, realizable by the partial automaton A with the input alphabet $\dot{X} = (x_1, x_2, \ldots, x_n)$ and with the output alphabet $\dot{Y} = (y_1, y_2, \ldots, y_m)$ uniquely determines the partition of the set F of all words in the alphabet $\dot{X}$ into m + 1 disjoint events in the alphabet $\dot{X}$, and namely into the events $R_1, R_2, \ldots, R_m$, represented in the automaton A by the output signals $y_1, y_2, \ldots, y_m$, and determines the forbidden domain S of the given (partial) automaton A.

And conversely: knowing the events $R_1, R_2, \ldots, R_m$ represented in some partial A by the output signals $y_1, y_2, \ldots, y_m$ we can uniquely recover the partial mapping φ between the words of the input alphabet $\dot{X}$ and the output alphabet $\dot{Y}$ realized by this automaton, without using the switching and output functions of the automaton.

Let there be given the arbitrary word $\ell = x_{i_1} x_{i_2} \ldots x_{i_n}$ in the alphabet $\dot{X}$. For each $k (1 \leq k \leq n)$ we find the output signal y_{j_k} using the rule: y_{j_1} is the output signal representing in the automaon A the event R_{j_k} which contains the initial segment $x_{i_1} x_{i_2} \ldots x_{i_k}$ of length k of the word ℓ. If for all $k = 1, 2, \ldots, n$ there exist the corresponding y_{j_k}, then we set $\varphi(\ell) = \varphi(x_{i_1} x_{i_2} \ldots x_{i_n}) = y_{j_1} y_{j_2} \ldots y_{j_n}$. In the case where an output signal $k = 1, 2, \ldots, n$ with the required properties does not exist for even one y_{j_k}, we assume that the partial mapping φ is not determinate on the word ℓ.

It is not difficult to see that as a result of the definition of events represented in an automaton, the partial mapping φ introduced in this fashion will then be precisely that partial mapping which is induced by the given partial automaton A.

On the basis of this discussion we can formulate the following proposition.

Theorem 2. The specification of the partial automaton mapping φ between words in the alphabets $\underset{\cdot}{X}$ and $\underset{\cdot}{Y} = (y_1, y_2, \ldots, y_m)$ is equivalent to the specification of the events $R_1, R_2, \ldots, R_m$ represented by the output signals $y_1\ y_2, \ldots, y_m$ in the partial automaton A which induces the mapping φ.

Theorem 2 lays the foundation for the study of the automaton mappings (in particular the automaton algorithms). For the description of such mappings it is sufficient to specify the partition of the set of all words of the input alphabet into a finite number of disjoint events. In order that the corresponding descriptions be of a constructive nature, it is necessary to limit ourselves to the consideration of only those events which admit effective description.

It is natural that first of all the finite events, i.e., events consisting of a finite number of words, admit simple constructive description. They can be described with the listing of the elements appearing in them. For the characterization of some important classes of infinite events, it is advisable to introduce several operations on the set of events, thus transforming this set into an algebra - the algebra of events.

For our purposes the most convenient is the system of three operations which is a modification of the operations first introduced by Kleene [40] (see also Copi, Elgot, Wright [45] and Glushkov [21]).

The first operation is that of the set-theoretic union of events. We shall designate this operation by the symbol $\vee$ and term it event disjunction.

The second operation is that of event multiplication, which is not to be confused with the operation of set-theoretic intersection. If the event S consists of the words $\ell_\alpha(\in M)$, and the event R consists of the words $q\beta(\beta \in N)$, then product of the events S and R is the name given to the event consisting of all possible words of the form $\ell\alpha q\beta$ ($\alpha \in M$, $\beta\in N$). The operation of event multiplication is non-commutative: generally speaking the events SR and RS are different.

The third operation is that of the so-called event iteration, for which we shall use the braces as the designation, so that {S} denotes the iteration of the event S. The iteration of any event S is defined as the union of an empty word, the event $S = S^1$ the event $S \cdot S = S^2$ the event $S \cdot SS = S^3$ and so on to infinity. In other words, if the event S consists of the words $\ell\alpha(\alpha\in M)$, then its iteration {S} consists of all possible words having the form $\ell\alpha_1 \ell\alpha_2 \ldots \ell\alpha_n$ where $\alpha_1, \alpha_2, \ldots,$ $\ldots, \alpha_n \in M$, and $n = 0, 1, 2, 3, \ldots$.

We shall term the braces used for the designation of iteration iteration brackets. For the designation of the order of operations we shall make use of round brackets, which we term conventional brackets. In the absence of brackets, used to alter the usual order of operations, iteration is to be performed first, then multiplication, and finally disjunction.

We agree to designate the single-element events, i.e., events consisting of a single word, by the symbol of this word. If $X = (x_1,$ $x_2, \ldots, x_n)$, then the m + 1 single-element events $x_1, x_2, \ldots, x_m, e$ are termed the elementary events in this alphabet.

Here and in the future we shall use the letter e to denote an empty word, consisting of an empty set of letters and consequently having zero length. This word will play only an auxiliary, service role. We agree, in particular, not to consider evnets which differ from one another only by an empty word as different. Thus, the empty word can, as desired, either be joined to or removed from any event in question. This is associated with the fact that as a result of the definitions which we have adopted the empty word cannot be represented in the automaton.

We shall now introduce a concept which is central to all the subsequent considerations.

Any event which can be obtained from the elementary events x_1, x_2, ..., x_m, e in the finite alphabet $\underset{\cdot}{X} = (x_1, x_2, \ldots, x_n)$ with the aid of the application of a finite number of operations of disjunction, multiplication and iteration is termed a regular event in this alphabet.

This definition goes back to the definition of the regular event given earlier by Kleene [40] although it differs considerably in form (see Glushkov [21]). We note that the same event can be represented differently in terms of the elementary events. In the future we shall term each such representation (formula of event algebra) a <u>regular expression</u>.

One of the primary problems in event algebra is the establishment of the laws of the <u>equivalent transformations</u> of the regular expressions, i.e., those transformations which do not change the events represented by these expressions (with an accuracy to the empty letter <u>e</u>).

Among the laws which are very frequently utilized in the equivalent transformations in event algebra are the laws of associativity for disjunction and multiplication, the commutativity law for disjunction, the left and right distributive laws for multiplication with respect to disjunction ($S(R \vee Q) = SR \vee SQ$, $(R \vee Q)S = (RS \vee QS)$ and others).

The laws of distributivity make possible, in particular, the removal of brackets and the bringing of common factors outside of the brackets (as in conventional algebra). Here we need only recall that multiplication in event algebra is generally speaking, not commutative.

Any word can be represented as the product of elementary events - the individual letters constituting this word. Any finite event is represented in the form of the disjunction of the words composing it. This implies, in particular, that all finite events are regular.

The use of iteration leads to the construction of infinite regular events. At the same time is is not difficult to construct simple examples of nonregular infinite events. For this it is sufficient to select such an increasing sequence of whole numbers n_1, n_2, ..., n_i, ..., that the differences $n_{i+1} - n_i (i = 1, 2, ...)$ are not bounded in the aggregate (this condition is satisfied, for instance, by the sequence of squares of the numbers of the natural series), and in any input alphabet $\underset{\cdot}{X}$ construct the event S consisting of all words in the alphabet $\underset{\cdot}{X}$ having lengths equal to n_1, n_2 and so on.

The event S constructed in this way is of necessity nonregular. Actually, assuming the opposite, we would be able to find for S some regular expression R. Since the event S is infinite, this expression contains at least one set of iteration brackets enclosing an expression differing from the empty word $\underline{e}$. Let us replace all the remaining iteration brackets in the expression R by an empty word, and the identified brackets by the expression {p} where $\underline{p}$ is an arbitrary nonempty word from the event enclosed in the identified brackets. As a result we obtain the regular expression R_1 for some event contained in the event S.

From the expression R_1 it follows directly that in the event S there appear words of the form rs, rps, rpps, rppps, ..., whose lengths constitute an infinite increasing arithmetic progression. But this contradicts the method of construction of the event S. Consequently, the event cannot be represented by any regular expression, i.e., it is a nonregular event.

Let us define also the concept of the cyclic depth of regular expression, meaning by this the maximal number of pairs of iteration brackets embedded in one another which are contained in this expression. For example, the expression $\{x\{y\}\{x\}\}$ has a cyclic depth of 2, while the expression $\{x \vee y\}x\{y\}$ has a cyclic depth of 1. By the cyclic depth of a regular event we shall understand the minimal cyclic depth of the regular expressions representing it.

Regular events have particular importance for the abstract theory of automata, since the class of regular events coincides with the class of events representable in finite automata. In the following sections we shall prove this important proposition; here we shall consider the question on the relationship of the classes of events representable in the Mealy and Moore automata.

The general definition of the representation of events in an automaton given in the beginning of the present section related to the Mealy automaton, Since the Moore automaton is a particular case of the Mealy automata, this definition is applicable in full measure to it as well. However, in practice it is convenient for the Moore automata to represent the events not by the property of the output at the instant of the application of the last input signal of the words comprising the events, but by the property of the state of the automaton after the arrival at the input of the automaton of a word of a particular event.

In other words, it is customary to consider that in the case of the Moore automata the events are some sets of the automaton states. On the strength of the definition of the Moore automata, this method of representing the events is completely equivalent to the method of representing the events by the sets of the output signals. The difference lies only in that with the representation of the events by the sets of the automaton states the empty word $\underline{e}$ is representable (with the aid of the initial state), while $\underline{e}$ cannot be represented by any output signal (if, of course, we do not initiate the time reckoning from negative instants of time).

However, we have agreed above not to consider events differing from one another only by the empty word $\underline{e}$ as different. Therefore the two methods of representation of events (states or output signals) in the case of the Moore automata are actually equivalent.

Since the Moore automata can be considered in the abstract theory as a particular case of the Mealy automata, it seems natural that the class of events represented in the Moore automata is more scanty than the class of events represented in the Mealy automata. In reality this is not so.

Let us assume that some event S is represented in some Mealy automaton A by the set M of its output signals. It is not difficult to see that the event S can be represented by some set of internal states of the Moore automaton B (inducing the same mapping φ as the automaton A) which was constructed in the preceding section.

We recall that the states of the automaton B are all possible pairs (a, x), composed from the states $\underline{a}$ of the automaton A and the letters $\underline{x}$ of its input alphabet X and also the initial state a_0 of the automaton A. The shifted output function μ of the automaton B on the initial state a_0 is determined arbitrarily, while on the state $b = (a, x)$ it is determined with the aid of the relation $\mu(b) = \lambda(a, x)$ where $\lambda(a, x)$ is the output function of automaton A.

If $h = gx_j$ is an arbitrary nonempty input word, then in the automaton A the last letter of the corresponding output word will obviously be $y = \lambda(a_0g, x_j)$. The automaton B, as it is not difficult to see, will be converted by the word $\underline{h}$ from the initial state a_0 into the state $(a_0 g, x_j)$.

Thus, all the nonempty words of the original event will be represented in automaton B by the set K of all possible states (a_i, x_j) for which the relation $\lambda(a_i, x_j) \in M$ is valid.

§3. ANALYSIS OF FINITE AUTOMATA

The analysis problem amounts to the determination of the events represented in the automaton by sets of output signals (in the case of the Mealy automata) or by sets of the internal states (in the case of the Moore automata). Since every Moore automaton can be interpreted as a Mealy automaton, it is sufficient to learn to analyze only the Mealy automata.

We shall resolve the analysis problem only for the case of finite Mealy automata. All events represented in such automata are necessarily regular. The analysis algorithm is applied to the switching and output tables of the automaton being analyzed and as the final information gives the regular expressions for the events representable by each of the output signals of the automaton. An event which is representable by an arbitrary set of output signals is written, then, as the disjunction of events represented by the individual output signals composing the given set.

Let us consider the arbitrary finite Mealy automaton A with the set of internal states $(a_1, a_2, \ldots, a_p)$ with the input alphabet $\underset{\cdot}{X} = (x_1, x_2, \ldots, x_n)$ and the output alphabet $\underset{\cdot}{Y} = (y_1, y_2, \ldots, y_m)$.

Considering specified the initial state a_1, the switching function $\delta(a_i, x_j)$ and the output function $\lambda(a_i, x_j)$ of the automaton A, we shall look for the regular expression R for the event represented by some output signal, say the signal y_1. We write out the internal states of the automaton $a_1, a_{j_1}, \ldots, a_{j_k}$ into which the automaton A transfers from the initial state a_1 by means of the sequential initial segments e, x_{i_1}, $x_{i_1} x_{i_2}$, ..., $x_{i_1} x_{i_2}$, ... x_{i_k} of some input word $\underline{q}$. Inserting the symbols for the states obtained into the word $\underline{q}$ after the corresponding initial segments, we transform this word into the new word $q' = a_1 x_{i_1} a_{j_1} x_{i_2} \ldots a_{j_{k-1}} x_{i_k} a_{j_k}$, which we agree to call the <u>path corresponding to the word $\underline{q}$</u>. Separating in the given path the symbols of the internal states a_j, we obtain the input word corresponding to the given path.

We shall also use the so-called <u>curtailed paths</u>, obtained from the conventional paths by the dropping of the extreme right symbol of the internal state a_{j_k}. We designate the path corresponding to the given input word $\underline{q}$ by $\underline{q}'$ and the curtailed path by $\underline{q}''$.

It is evident that for the nonempty input word $\underline{q}$ to belong to the event R_i, representable in the automaton A by the output signal y_i, it is necessary and sufficient that the curtailed path $\underline{q}'$ corresponding to the word $\underline{q}$ terminate with the pair $a_{j_{k-1}} x_{i_k}$, for which the output function takes the value equal to y_i. We term all such (curtailed) <u>paths of the type</u> y_i, or, generalizing (for any $\underline{i}$), representative type paths.

Paths (uncurtailed) corresponding to the input words which transform the automaton from some state a_j into the same state a_j are termed <u>type</u> a_j <u>paths</u>, or <u>cyclic type paths</u>. If in some path $\underline{q}'$ of the cyclic type a_j there are no symbols of any internal states a_{k_1}, a_{k_2}, ..., a_{k_r} then we shall also term the path $\underline{q}'$ a path of the $a_j[a_{k_1}$, a_{k_2}, ..., $a_{k_r}]$ type (here the symbols in the square brackets are termed forbidden).

The path $\underline{q}'$ of arbitrary type is termed <u>simple</u> if the curtailed path $\underline{q}''$ corresponding to it does not contain two identical symbols of the internal states. Only a finite number of different simple paths exists in a finite automaton. All simple paths of any given type can be found directly from the switching table or (for paths of the representative type) from the switching and output tables of the automaton.

Let us construct some auxiliary events in the alphabet $Z = (x_1, x_2, \ldots, x_n, a_1, a_2, \ldots, a_p)$, whose elements are curtailed paths in the given automaton A. We define the event $S(y_1)$ of type y_1 as an event consisting of all (curtailed) paths of type y_1, we define the simple event $P(y_1)$ of type y_1 as the disjunction of all simple (curtailed) paths of type y_1. We shall term the iteration of the disjunction of all simple paths of type $\underline{t}$ the simple event $P(t)$ of any given cyclic type $t = a_j[a_{k_1}, a_{k_2}, \ldots, a_{k_r}]$ $(j = 1, 2, \ldots, p,\ r \leq p - 1)$ (the disjunction of an empty set of paths is an impossible event whose iteration coincides with the empty word $\underline{e}$) and shall term the event consisting of all curtailed paths of type $\underline{t}$ the event $S(t)$ of type $\underline{t}$. Finally, we term the iteration of the portion of the event $S(t)$ containing words with only a single occurrence of the symbol a_j the <u>conditionally simple event</u> $U(t)$ of type $t = a_j[a_{k_1}, a_{k_2} \ldots, a_{k_r}]$.

Let there be given some set (curtailed) of paths of type y_1 specified with the aid of the regular expression Q. Inserting into this regular expression ahead of each occurrence in it of the symbol of the internal state a_j the regular expression of the event $S(a_j)$ of type a_j or of the event of type $a_j[a_{k_1}, a_{k_2}, \ldots, a_{k_r}] = t$, we obtain a new regular expression, representing as before only paths of type y_1. We term this operation the <u>embedding</u> of the event $S(a_j)$ in the event Q.

Now let $\underline{q}''$ be an arbitrary (curtailed) path of type y_1. The first (left) symbol of the internal state occurring in this path will be the symbol a_1. Let us isolate also the last (extreme right) occurrence of the symbol a_1 in the path q": q" = $a_1 \ldots a_1 s$, where the word s already does not contain the symbol a_1. Then the path q" can be represented by the product of some number of words of the conditionally simple event $U(a_1)$ of type a_1 and the word $a_1 s$.

In the word s we find the first (left) symbol of the internal state: $s = x_{i_k} a_{j_k} \ldots$; after finding also the last occurrence of this symbol in the word s, we obtain the possibility of representing the word s by the product of the letter x_{i_k}, some number of words of the conditionally simple event of type $a_{j_k}[a_1]$ and some word $a_{j_k} r$ where r does not contain the two symbols of the internal states a_1 and a_{j_k}.

We further come to the conclusion that the path q" is contained in the event which is obtained as the result of the embedding of the conditionally simple event of type a_1 in some simple path of type y_1 ahead of the first occurrence in it of the letter a_1, and the embedding of the conditionally simple events of type $a_{j_k}[a_1]$, $a_{j_e}[a_1, a_{j_k}]$ ahead of the succeeding occurrences in this path of the symbols of the internal states a_{j_k}, a_{j_e}, ..., and so on.

But exactly the same process, obviously, can be repeated with the words of the conditionally simple events which were separated from the original path q". After this we come to the conclusion that the path q" of type y_1 occurs in the event which is obtained as the result of the embedding in the simple event $P(y_1)$ of type y_1 not the conditionally simple, but the ordinary simple events of types a_1, $a_{j_k}[a_1]$, ... and the subsequent embedding in the paths constituting the embedded simple events of the conditionally simple events: for the conditionally simple event of type a_1 – the types $a_x[a_1]$, $a_y[a_1, a_x]$, ..., for the simple event of type $a_{j_k}[a_1]$ – the types $a_u[a_1, a_{j_k}]$, $a_v[a_1 a_{j_k} a_u]$,... and so on.

We further come to the conclusion that again in the second stage we can embed not the conditionally simple, but the ordinary simple events, embedding, in turn (in the third stage) in the words constituting them the conditionally simple events of still higher (in the sense of the number of forbidden letters) cyclic types.

Increasing the number of stages of sequential embeddings, we finally come to the embedding of events of cyclic types in which all the letters except one are forbidden. Since for this kind of types the difference between the conditionally simple and the ordinary simple events of identical type no longer exists, the process of increasing the number of stages and of new embeddings is thereby completed.

As a result we come to the conclusion that the path q" occurs in the event S_1 which is obtained as the result of a finite number of sequential embeddings (divided into a number of stages) into the simple event of type y_1 of simple events of ever higher and higher cyclic types. In view of the arbitratiness of the selection of the path q", the event S_1 includes in itself the event $S(y_1)$.

At the same time, as remarked above, the process of embeddings similar to the process described cannot lead to an event containing the paths differing from the y_1 type. Consequently, $S_1 = S(y_1)$, and the embedding process we have described gives a regular expression R_1 for the event $S(y_1)$, consisting of all paths of type y_1.

Dropping now in the regular expression R_1 all the symbols of the internal states (replacing them with an empty word), we obtain the regular expression R which, as it is easy to see, is nothing other that the regular expression for the sought event, represented in the automaton A by the output signal y_1.

We have proved the following proposition.

Theorem 1. An event represented in an arbitrary finite Mealy automaton (and, consequently, also in an arbitrary finite Moore automaton) by any set of output signals is necessarily regular. There exist a universal constructive technique (algorithm for the analysis of finite automata) which makes it possible to find the regular expressions for events represented by the sets of output signals in an arbitrary finite automaton.

The described algorithm for the analysis of finite automata can be given a form which is more convenient for practical applications [22]. To do this we shall work not with the events in the set of paths, but with formal expressions termed complexes.

For any set M of output signals of a given finite Mealy automaton A the term complex of type M (or output type complex) is given to the disjunction of all simple curtailed paths terminating with the pairs $a_j x_i$, to which there correspond the output signals contained in the set M. Complex of type $a_i[a_{i_1}, a_{i_2}, \ldots, a_{i_r})$ (cyclic type complex) is the term for the formal expression obtained as the result of joining with the disjunction sign all simple paths of type $a_i[a_{i_1}, a_{i_2}, \ldots,$ $\ldots, a_{i_r}]$ with the letter a_i stricken and enclosing the resulting formal polynomial in the iteration brackets ($a_i, a_{i_1}, \ldots, a_{i_r}$ are any pairwise different internal states of the automaton and $0 \leq r \leq p - 1$ where p is the number of internal states of the automaton A).

First step of the analysis algorithm. From the switching and output tables of the automaton and the given (representative) set of output signals, by means of sorting of all possible variants of simple paths we find the complex K(M) of type M and the complexes $K(a_i)$ for all the internal states a_i of the automaton A.

Second step. From the complexes $K(a_i)$, by exclusion of unnecessary terms in the iteration brackets we find the complexes of higher cyclic types $a_i[a_{i_1}, a_{i_2}, \ldots, a_{i_r}]$ $(r \leq 1)$. which are necessary for the further constructions.

Third step. Starting from the complex of type M, we sequentially replace all symbols of the internal states a_i by complexes of cyclic type until we obtain an expression R not containing a single one of the internal state symbols. The replacement rule can be formulated as:

If the path $a_1 x_{i_1} a_{j_1} x_{i_2} a_{j_2}$ ... occurs in the complex of the type M output, then the letter a_1 is replaced by the complex of type a_1, the letter a_{j_1} is replaced by a complex of the type $a_{j_1}[a_1]$, the letter a_{j_2} by a complex of the type $a_{j_2}[a_1, a_{j_1}]$ and so on. If the term $x_{i_1} a_{j_1} x_{i_2} a_{j_2}$ ... occurs in the complex of the type a_i [N], where N is the set (possible empty) of internal states differing from a_1, then letter a_{j_1} is replaced by a complex of the type $a_{j_1}[a_1, N]$, the letter a_{j_2} by the complex of the type $a_{j_2}[a_1, a_j, N]$ and so on.

In the third step, as a result of the application of the replacement rule a finite number of times, we obtain the desired regular expression R for the event represented in the automaton A by the set M of output signals.

The following proposition follows directly from the described algorithm.

Theorem 2. Every event represented in a finite Mealy automaton (or Moore automaton) having n internal states admits a regular expression whose cyclic depth not does exceed n.

As an example let us find the regular expression for the event S represented by the output signal v in the automaton whose switching and output tables were described in §1 of the present chapter (its graph is shown in Fig. 7).

We find the complex K(v) of type v directly from the tables

$$K(v) = {}_1y \vee {}_1y_3x \vee {}_1x_2x_3x$$

and the complexes of types 1, 2 and 3

$$K(1) = e,\ K(2) = \{y \vee x_3 y\},\ K(3) = \{x \vee y_2 x\}.$$

We write out some complexes of the higher cyclic types:

$$K(2[1]) = K(2);\ K(3[1,2]) = \{x\};$$
$$K(2[3]) = K(2[1,3]) = \{y\};\ K(3[1]) = K(3).$$

Designating the operation of embedding of complexes by an arrow, we obtain the following sequence of embeddings:

$$\begin{gathered} K(v) \rightarrow y \vee yK(3[1])x \vee xK(2[1])xK(3[1,2])x = \\ = y \vee y\{x \vee y_2 x\}x \vee x\{y \vee x_3 y\}x\{x\}x \rightarrow y \vee y\{x \vee \\ \vee yK(2[1,3])x\}x \vee x\{y \vee xK(3[1,2])y\}x\{x\}x = \\ = y \vee y\{x \vee y\{y\}x\}x \vee x\{y \vee x\{x\}y\}x\{x\}x. \end{gathered}$$

The last of the regular expressions obtained is then the sought regular expression for the event S. It admits transformation and simplification with the use of the relations existing in event algebra.

§4. ABSTRACT SYNTHESIS OF FINITE AUTOMATA

The abstract synthesis problem is the opposite of that of the analysis of finite automata: it is necessary to find an effective method which will make it possible to find from the regular expressions for the events the switching and output tables of some finite automaton which represents these events.

The problem of the synthesis of Moore automata is more general than that of the synthesis of the Mealy automata: since every Moore automaton can be interpreted as a Mealy automaton, by learning to synthesize the Moore automaton we also learn to synthesize the Mealy automaton as well. Therefore we shall solve the problem of synthesis of the Moore automaton.

Let there be given in some finite alphabet $\underset{\cdot}{X} = (x_1, x_2, \ldots, x_n)$ the p regular expressions $R_1, R_2, \ldots, R_p$. Let us number all occurrences of the letters of the alphabet $\underset{\cdot}{X}$ in the expressions $R_1, R_2, \ldots,$ $\ldots, R_p$ by the sequential natural numbers, which hereafter we shall term the subscripts of the corresponding places of these expressions. We emphasize particularly that the various occurrences of the same letter of the alphabet $\underset{\bullet}{X}$ will thus have different subscripts.

In the development of the regular expression into a word, each of the sequentially written out letters of this word is identified with a particular occurrence of the corresponding letter in the expression being developed. We agree to consider that in this identification we enter particular places of the regular expression, namely: in the identification of the last written letter with the occurrence numbered with the subscript j, we shall consider that we are in the j-th place of the corresponding regular expression. We say that the j-th place of the regular expression x_k-follows after the i-th place if after identification of the last letter of some word q with the occurrence having the subscript i we can identify the last letter of the word qx_k with the occurrence having the subscript j. In each regular expression there is also identified the initial place, to which there is assigned the subscript 0 (identical for all given regular expressions). If, in the process of identification, the first letter x_k of some word is identified with some occurrence of it in the regular expression, having the subscript j, then we consider that the j-th place x_k-follows after the zero (initial) place (common for all given regular expressions).

Finally, if the membership of some word p to the event with the regular expression R is established as the result of the identification of the last letter of the word p with its occurrence in R, having the subscript j, then the j-th place of the expression R is termed a final place of this expression.

For any finite set of regular expressions $R_1, R_2, \ldots, R_p$ in the same alphabet $\underset{\bullet}{X}$, using the order of operations in the algebra of events defined above, it is not difficult to compose the place sequence table. The rows of this table are designated by the letters of the alphabet $\dot{X}$ and the columns by the subscripts of all the places of the expressions R_1, R_2, ..., R_p. At the intersection of the x_i-th row with the j-th column of the sequence table there are written out the subscripts of all the places which x_i-follow after the j-th place. If there are no such subscripts, in the corresponding place of the table we place a special symbol designating an empty set of subscripts. We agree to use an asterisk as this symbol.

Let us construct the Moore automaton A whose internal states will be all possible subsets of place subscripts in the given regular expressions R_1, R_2, ..., R_p (including the empty subset). The switching function δ of this automaton is constructed as follows: for any state a_i of the automaton A (set of place subscripts of the given events) and for any letter x_j of the input alphabet, the state $a_k = \delta(a_i, x_j)$ is defined as the set of subscripts of all places which x_j-follow at least one of the places whose subscripts occur in a_i.

The shifted output function μ of the Moore automaton A is constructed for the output alphabet $\dot{Y}$ consisting of all possible subsets (including the empty subset) of the set of all symbols R_1, R_2, ..., R_p of the given regular expressions. For any state a_i (set of subscripts) of the automaton A, we select as $\mu(a_i)$ the set of all those regular expressions R_1, R_2, ..., R_p, for which at least one of the subscripts occurring in a_i is the subscript of a final place.

We have constructed some finite Moore automaton A. From the method of construction of its switching and output functions it follows directly that it represents (with selection of 0 as the initial state) each of the given events R_1, R_2, ..., R_p and the complement S of their union. The event R_i is represented by the set of all those output signals (sets of symbols R_1, R_2, ..., R_p), in whose composition there occurs the symbol $R_i (i = 1, 2, ..., p)$. The event S is obviously the empty set of the symbols R_1, R_2, ..., R_p.

As a result we have proved the following proposition.

Theorem 1. Any regular event can be represented in a finite automaton. There exists a single constructive technique (synthesis algorithm) which makes it possible from any finite set of regular events given by regular expressions to construct the finite Moore or Mealy automata representing these events.

Combining the proved theorem with the result obtained in §2, we obtain the following result.

Theorem 2. Regular events and only regular events are representable in finite automata.

A similar result for automata of a special form (neural networks) and for a more awkward form of definition of the regular event has been obtained previously by Kleene [40].

The following proposition also follows from the results obtained above.

Theorem 3. The intersection of two (and therefore of any finite number as well) regular events and the complement (in the set of all words in the basic alphabet) of any regular event are also regular events.

The algorithm described above for the synthesis of finite automata also admits the following interpretation which is more convenient for practical purposes [21].

Let there be given the p regular expressions R_1, R_2, ..., R_p in the arbitrary finite alphabet $X = (x_1, x_2, ..., x_n)$. If any of the expressions R_i is the disjunction of several terms, then we can without losing generality consider that it is enclosed in ordinary (noniterative) brackets. Specially introduced separation symbols (vertical bars) standing between any two symbols (letters, brackets, disjunction signs) of these expressions and also standing to the left of an expression (initial place) and to the right of an expression (final place) will be termed <u>places</u> in the expressions R_1, R_2, ..., R_p.

Places having a letter of the basic alphabet $\dot{X}$ standing directly on their left and the initial place are termed basic places; the places having a letter of the alphabet $\dot{X}$ standing directly on their right are termed prebasic. The initial places of all the expressions R_1, R_2, ..., ..., R_p are identified with one another in one single initial place. We designate all the basic places with different nonnegative whole numbers — the basic subscripts of these places. Here the initial place takes the basic subscript 0.

The operation of each basic subscript extends not only to the corresponding place, but also to the places (basic and nonbasic) which are subordinate to it. The place subordination rule expresses the order of the operations in the algebra of events. It is defined by the following <u>subscript extension rule</u>.

The place subscripts ahead of any brackets (iterative or conventional) extend to the initial places of all the terms standing inside these brackets. The subscripts of the final place of any term enclosed in brackets extend to the place directly following these brackets. Place subscripts directly preceding iterative brackets or symbols of an empty word extend to the place directly following these brackets (respectively after the given symbol $\underline{e}$). Finally, place subscripts following directly after iterative brackets extend to the initial places of all the terms enclosed in these brackets.

All the subscripts appearing on the basic and nonbasic places as the result of the application of the rule just formulated are termed nonbasic. In this case the rule itself must be applied until its application no longer leads to the appearance of now subscripts on any place.

The indexing of the given regular expressions, the labeling of the places and the extension of the subscripts according to the formulated rule constitute the first step of the synthesis algorithm.

The second step consists in the construction of the switching table of the sought automaton A. Here the input signals are the letters of the original alphabet X, and the internal states of the automaton are identified with the sets of the basic subscripts. Let us agree for definiteness to denote these sets by the disjunction of the component subscripts, and the empty set of subscripts by an asterisk.

The rule for the construction of the automaton switching table amounts to the following.

The single-element set consisting of the subscript 0 serves as the initial state of the automaton A. The state a_1 is transformed by the input signal x_k into the state a_j, consisting of the basic subscripts of all the basic places, separated by the letter x_k from the prebasic places directly preceding them, whose subscripts (basic or nonbasic) contain at least one subscript from the number of subscripts occurring in the state a_1.

In practical application of the formulated rule it is convenient to separate the basic subscripts, placing them above a horizontal line specially drawn for this purpose. It is also advisable to separate all the subscripts (basic and nonbasic) of the prebasic places, for example enclosing them in a rectangular frame. In the construction of the switching table it is sufficient to limit ourselves to only the states which actulaly appear in the process of the construction of the table, starting from the initial (zero) state.

The third step of the synthesis consists in the construction of the shifted output table, or, what is the same, in the labeling of the states of the automaton A with the output signals corresponding to them. As the output signals we select the various sets of the symbols of the initial regular expressions (including the empty set). The state labeling rule consists in the following.

The state a_i is labeled with the set of those symbols of the expressions $R_1, R_2, \ldots, R_p$, whose final place subscripts (basic and nonbasic) include at least one subscript from a_i.

The states labeled with the empty set of symbols are also termed unlabeled.

We note that the constructed Moore automaton represents the event R_i by the set of all those output signals which contain the symbol R_i $(i = 1, 2, \ldots, p)$.

The fourth step of the synthesis algorithm consists in the redesignation of the internal states and the output signals to obtain a simpler writing of the switching table and the shifted output table. Here the internal states are most frequently numbered with the sequential natural numbers 1, 2, ..., k.

Finally, the fifth step of the synthesis algorithm is used when we are required to synthesize a Mealy automaton rather than a Moore automaton. It amounts to the construction of the conventional (unshifted) output table. As follows from §1 of the present chapter, for this it is sufficient to substitute in the switching table in place of the internal states the output signals which label them.

In the solution of practical problems which arise in the synthesis of automata, it is frequently convenient to assign identical basic subscripts to certain basic places, thereby identifying these places. Such an identification is possible if to the identified places there are subordinated identical sets of prebasic and final places (places satisfying this conditions are termed similar).

Another case when it is possible to identify places relates to the so-called corresponding places. All those places in the various regular expressions R_1, R_2, ..., R_p or in the different terms enclosed in the same brackets, to which identical paths (sets of words) lead from the initial place or correspondingly from the place directly preceding the brackets, are termed corresponding.

In the use of the synthesis algorithm described above the basic subscripts of the corresponding places always occur together in the states of the automaton being synthesized. It is precisely this that makes possible their identical indexing. Substantiation of the possibility of identifying similar places results from the minimization algorithm described in §5 of the present chapter.

We note that the places should be identified only with respect to one of the criteria (similarity or correspondence), since simultaneous identification with respect to both criteria can lead to errors. In particular, since the initial places are actually identified with respect to the correspondence criterion, we cannot, generally speaking, with the existence of more than one event identify the initial place in any event with another place using the similarity criterion.

The validity of the following proposition results directly from the algorithm described.

Theorem 4. Events given by the regular expressions R_1, R_2, ..., R_p in some finite alphabet $\dot{X}$ can be represented in a finite automaton (Mealy or Moore) having no more than 2^{n+1} internal states, where n is the total number of occurrences of the letters of the alphabet $\dot{X}$ in the expressions R_1, R_2, ..., R_p.

Let us consider what changes need to be made in the synthesis algorithm when, in addition to the initial events R_1, R_2, ..., R_p, there is also given the forbidden region S in the alphabet $X = (x_1, x_2, \ldots, x_m)$.

The forbidden region can be specified either with the aid of some regular expression, or as an ensemble of words in the alphabet X not occurring in even one of the events R_1, R_2, ..., R_p. These two methods are essentially equivalent to one another, since we can transfer from the first method of specification to theother and vice versa.

The forbidden region S by its very definition permits right multiplication by the ensemble F of all words (including the empty word) in the alphabet X: SF = S. Therefore, with specification of the forbidden region by the regular expression R we can, without losing generality, assume that the expression R has the form

$$R = R_1\{x_1 \vee x_2 \vee \ldots \vee x_n\}.$$

The synthesis algorithm described above gives the solution of the problem with the existence of a forbidden region. However, in this case many transitions in ther synthesized automaton are redundant in the sense that they will never be used in actual operation of the automaton. The problem consists in the determination of all such switchings and the construction in place of the conventional (completely determinate) automaton a partial automaton in whose switching and output tables dashes stand in the places of the forbidden transitions. The conversion to the partial automaton gives additional possibilities for subsequent simplification of the automaton.

This problem in the case of the specification of the forbidden region as the ensemble of words in the alphabet X which do not occur in even one of the given regular expressions R_1, R_2, ..., R_p, is solved by a quite obvious method. After the performance of the synthesis algorithm described above, the output signal designated by the empty set of symbols R_1, R_2, ..., R_p, will correspond to the appearance of the forbidden input word. Consequently, it is sufficient to replace this output signal in the output table by a dash and to put a dash in all the places of the switching table corresponding to the appearance of the forbidden output signal (with superpositioning of the output table on the switching table the places labeled with a dash in the two tables must coincide).

In the case of the specification of the forbidden region by the regular expression S, we apply the usual synthesis algorithm to the expressions S, R_1, R_2, ..., R_p and consider as forbidden all outputs designated by the sets which include the symbol S. Forbidden outputs in the output table are replaced by dashes, which are transferred to the switching table using the method described above. It is clear that this technique actually leads to the solution of the posed problem.

In this case we should consider that the expression has the form

$$S = S_1 \{x_1 \vee x_2 \vee \ldots \vee x_n\}.$$

If the initially given expression for the forbidden region did not satisfy this condition, it must be replaced by the expression

$$R = S \{x_1 \vee x_2 \vee \ldots \vee x_n\}.$$

As an example, let us consider the synthesis of the partial Mealy automaton representing the event R = x {y}, with the existence of the forbidden region $S = yx\{x \vee y\}$. In the first step we perform the labeling of the places, the indexing and the extension of the indexes in the expressions R and S, using the possibility of identification of similar places:

$$\begin{array}{cccccccccc} R = & | & x & | & \{ & | & y & | & \} & | \\ & \overline{0} & & 1 & & & & 1 & & \\ \hline & & & & & \underline{\overline{1}} & & & & 1 \end{array} \qquad \begin{array}{cccccccccccccccc} S = & | & y & | & x & | & \{ & | & x & | & \vee & | & y & | & \} & | \\ & \overline{0} & & \overline{2} & & 3 & & & & 3 & & & & 3 & & \\ \hline & & & & & & & \underline{\overline{3}} & & & & \underline{\overline{3}} & & & & 3 \end{array}.$$

Performing the second and third steps of the algorithm, we come to the labeled switching table of the Moore automaton

	—	*R*	—	—	*S*
	0	1	2	*	3
x	1	*	3	*	3
y	2	1	*	*	3

.

In the fourth step we introduce the redesignation 0 → 1, 1 → 2, 2 → 3, * → 4, 3 → 5, ( ) → u, R → v, S → w (here the brackets ( ) designate the empty set of symbols R and S). After this obtain the labeled switching table

	u	*v*	*u*	*u*	*w*
	1	2	3	4	5
x	2	4	5	4	5
y	3	2	4	4	5

.

Completing in the fifth step the conversion to the Mealy automaton, we obtain the switching and output tables

	1	2	3	4	5
x	2	4	5	4	5
y	3	2	4	4	5

;

	1	2	3	4	5
x	*v*	*u*	*w*	*u*	*w*
y	*u*	*v*	*u*	*u*	*w*

.

Finally, we perform the conversion to the partial automaton. In the present case we shall consider the forbidden region to be the signal <u>w</u>. Then the switching and output tables will have the form

	1	2	3	4	5
x	2	4	—	4	—
y	3	2	4	4	—

;

	1	2	3	4	5
x	*v*	*u*	—	*u*	—
y	*u*	*v*	*u*	*u*	—

.

However, state 5 is redundant, since the automaton can never convert into it starting from the initial state 1. Discarding this redundant state, we come to the final switching and output tables

	1	2	3	4
x	2	4	—	4
y	3	2	4	4

;

	1	2	3	4
x	*v*	*u*	—	*u*
y	*u*	*v*	*u*	*u*

.

§5. MINIMIZATION OF ABSTRACT AUTOMATA

As indicated above, we consider the abstract automaton as a device for the realization of automaton mappings. In connection with this it is natural not to differentiate automata which are equivalent to one another, i.e., automata which induce identical mappings.

The primary task resolved in the present section is that of the minimization of automata, i.e., the problem of finding the automaton with the minimal number of states in the class of all automata equivalent to the given one. The method presented for the solution of this problem is a development of the ideas of Mealy [55], Aufenkamp and Hohn [3,4].

Let a and b be two states of the same or of two different Mealy automata having common input and output alphabets. If for any input signal x_1 the output signals determined by the pairs (a,x_1) and (b,x_1) are identical, then the states a and b are termed 1-equivalent states.

If 1-equivalent states are transformed by any input signal x_1 into states which also are 1-equivalent to one another, then they are termed 2-equivalent. If 2-equivalent states are transformed by any input signal into states which are 2-equivalent to one another, then they are termed 3-equivalent, etc.

It is easy to see that in the case of the application to them of any input word i the i-equivalent states give rise to identical output words i = 1, 2, ...).

The i-equivalency relation for any i = 1, 2, ... has the properties of reflexivity, symmetricity and transitivity. This implies that the set of all internal states of a given Mealy automaton is partitioned by this relation into disjoint classes of states which are i-equivalent to one another. We term such classes i-equivalent classes or i-classes.

States which are i-equivalent for all i = 1, 2, ..., are termed equivalent states, and the classes defined by the equivalency ratio are termed equivalent classes or ∞-classes.

The validity of the following proposition follows directly from the definition of the states which are equivalent to one another.

Theorem 1. Two states of the same or of two different Mealy automata are equivalent to one another if and only if the application to them of any input word causes the appearance of the same output word.

This proposition makes it possible to formulate the following result as well.

Theorem 2. Two Mealy automata are equivalent to one another (in the sense of the coincidence of the automaton representations which they induce) if and only if their initial states are equivalent.

The application of the same input word p to two equivalent states a and b transforms them anew into the equivalent states ap and bp. Since equivalent states are at the same time 1-equivalent, then for any input signal x_i the pairs (a, x_i) and (b, x_i) define identical output signals.

Thus, for every Mealy automaton A we can construct the new Mealy automaton B with the same input and output alphabets as automaton A, taking as the set of its internal states the set of all equivalence classes of the automaton A. The transitions and the outputs in automaton B are determined as follows: the equivalence class K_1 is transformed by the input signal x_i into the equivalence class K_2 containing the state $a_j x_i$, where a_j is any state contained in the class K_1. To the pair $K_1 x_i$ there is associated in this case the output signal determined by the pair $a_j x_i$. We shall term the automaton thus constructed the canonical minimization of the Mealy automaton A.

The validity of the following proposition follows from the method of construction of automaton B and from proposition 1.

Theorem 3. In the canonical minimization of any Mealy automaton, any two different internal states are not equivalent to one another.

For the realization of automaton correspondence it is sufficient to limit ourselves to the consideration only of the so-called <u>connected</u> automata, i.e., those automata in which every state is attainable, or, in other words, which as the result of the application of a suitable input word can be transformed from the initial state into any other internal state.

Actually, if the mapping φ is induced by the unconnected automaton A, then the attainable states of the automaton A form the new automaton B which is connected and induces the same mapping φ. We note that as a result of the application of the synthesis algorithm described in the preceding section, connected automata are always obtained.

Let us consider some connected Mealy automaton A which induces the specified automaton mapping φ. The canonical minimization B of the automaton A is also connected and realizes the same mapping φ (with selection as the initial state of the equivalence class K_0 containing the initial state of the automaton A).

Let D be any automaton realizing the same correspondence and let d_0 be its initial state. On the strength of the connectedness of the automaton B, for its every state K_i we can select the input word p_i such that $K_0 p_i = K_i$ $(i \in M)$. Let us construct the mapping ψ of the set of states of automaton B into the set of states of automaton D, setting $\psi(K_i) = d_0 p_i$ $(i \in M)$.

It is clear that the initial states K_0 and d_0 of the automata B of the automata B and D are equivalent to one another. But then the states K_i and $d_i = \psi(K_i)$ are also equivalent for any $i \in M$. If $\psi(K_i) = \psi(K_j)$, then this implies equivalence of the states K_i and K_j. As the result of proposition 3 this means that $K_i = K_j$. Thus, the correspondence ψ is one-to-one, which implies the validity of the following proposition.

Theorem 4. The canonical minimization of a connected Mealy automaton which induces any given automaton mapping φ is the automaton having the smallest possible number of internal states along all Mealy automata which induce the same mapping φ (i.e., among all automata equivalent to the automaton A).

This statement completely resolves the problem of the minimization of the Mealy automata under the condition that there exists the constructive technique for the construction of the equivalency classes for any given (connected) Mealy automaton. Such a technique has been suggested by Aufenkamp and Hohn [4] for the scale of finite Mealy automata. It is based on the following easily proved proposition.

Theorem 5. If for some i the partition of the states of the automaton into $(i + 1)$-classes coincides with the partition into i-classes, then it is also the partition into ∞-classes.

Actually, if any pair (a_k, a_j) of i-equivalent states is also $(i + 1)$-equivalent, then the states a_k and a_j are transformed by any input signal x_r into states which are i-equivalent to one another. But then they are transformed by this same signal also into states which are $(i + 1)$-equivalent to one another. Consequently, the states a_k and a_j are not only $(i + 1)$-equivalent, but are also $(i + 2)$-equivalent to one another.

We further find that the states a_j and a_k are n-equivalent for all $n = i, i + 1, i + 2, \ldots$ and, consequently, are equivalent states. In view of the arbitrariness of the choice of the states a_j and a_k, proposition 5 is proved.

The Aufenkamp-Hohn algorithm for the construction of the equivalence class (∞-classes) is based on the sequential construction of i-classes for all $i = 1, 2, \ldots$. Since the partition into $(i + 1)$-

classes is a subpartition of the partition into i-classes, then in the case of finiteness of the automaton A, after a finite number of steps we obtain on the basis of theorem 5 the sought partition into ∞-classes.

The partition into 1-classes is performed directly from the output table of the automaton: into the same 1-class there are combined all the states to which there correspond identical columns in the output table. Then there is constructed the so-called 1-table, obtained as the result of replacement in the automaton switching table of the internal states by the 1-classes which contain them.

In a single 2-class there are combined all the states belonging to the same 1-class to which there correspond identical columns in the 1-table. Then we proceed similarly: replacing in the switching table the automaton states by the 2-classes which contain them, we obtain the 2-table. From the 2-table we find the 3-classes, combining in one 3-class all the states of the same 2-class to which there correspond identical columns in the 2-table.

Arriving after a finite number of steps at the partition into ∞-classes, we construct the canonical minimization of the original automaton A directly from its switching and output tables.

As a result we have constructed the minimization algorithm for any finite Mealy automata. For the case of the Moore automata it is necessary to introduce certain changes in this algorithm, since by interpreting the Moore automaton as a Mealy automaton and minimizing it in accordance with the described algorithm, we construct an automaton which, although equivalent to the original, is possibly not now a Moore automaton.

In order that the Moore automaton remain a Moore automaton during minimization it is evidently necessary and sufficient that identically labeled states of the automaton not be related to different equivalency classe

This [illegible]tion can be satisfied most simply by introducing for the Moore automaton the concept of <u>0-equivalency</u> of the states and the partition of the set of states into <u>0-classes</u>: we shall term any <u>identically</u> labeled states of a Moore automaton 0-equivalent. If two 0-equivalent states are transformed into two 0-equivalent states by any input signal, then they are termed 1-equivalent.

All the further constructions (determination of the i-classes for $i \geq 2$, determination of the equivalency classes and the construction of the canonical minimization) are performed just as in the case of the Mealy automata. Of course, in the case of the Moore automata for the construction of the canonical minimization B we can specify for it not the output table, but the <u>shifted</u> output table, labeling the states of the states of the automaton B (equivalence classes) by the same output signals which are used to label the states of the original automaton which occur in it.

However, the theory of minimization of Moore automata in the form just described is not fully equivalent to the corresponding theory for the case of the Mealy automata. In particular, the proposition analogous to theorem 1 does not extend to the Moore automata.

To obtain equivalence of the two theories it is necessary to consider as the reaction of the Moore automaton to the input word <u>p</u> not that output word <u>q</u> which is obtained as the result of the general definition of the automaton given in §1, but the word $y_i q$, where y_i is the output signal labeling that state in which the automaton was prior to the application of the word <u>p</u>. With this definition of the reaction of the Moore automaton to the input word for this automaton, the propositions obtained from theorems 1, 2, 3, 4 of the present section by replacement of all Mealy automata encountered in their formulation by

Moore automata will evidently be valid. Theorem 5 remains, of course, valid for the Moore automata with the usual definition of the output reactions of the automata.

With conversion to the usual understanding of the output reactions of the automaton, only those states into which the automaton cannot transfer starting from the other states are found to be in a special situation (for the case of connected automata only the initial state can have this property). It is easy to verify that for the restoration of parallelism in thetheories of the minimization of the Moore and Mealy automata, in this case it is sufficient not to label similar states with any output signals. This permits in the formation of the O-classes relating such states to any O-class and thereby increases the possibilities of the minimization.

However, the parallelism appearing here takes place not with minimization of the conventional everywhere-determinate automata, but with transfer to the more general problem of the minimization of partial automata.

The essence of the problem of the minimization of the partial automata amounts to the following: given the partial automata (Moore or Mealy) A which induces the partial automaton on mapping φ defined on some set M of words of the input alphabet. Required to construct the partial automaton (Moore or Mealy respectively) B which induces the partial automaton mapping coinciding on the set M with the mapping φ and which has the smallest number of internal states among all automata (Moore or Mealy) satisfying this condition.

Since there is only a finite number of differnt partial automata in which the input and output alphabets are common with the given finite partial automaton A and in which the number of states does not exceed the number of states of automaton A, the formulated problem is algorithmically solvable. However, the existing methods for the exact solution of this problem are associated with extensive sorting (see, for example, Ginsburg [19]) and are therefore unsuitable in practice.

In practice, use is usually made of the following technique for the minimization of the partial automata: mentally filling in the stricken places in the switching and output tables of the given partial automaton A, we combine the states into k-classes and minimize using the same rules as in the case of conventional (everywhere determination of states into classes are checked, and from the resulting canonical m minimizations we select that which has the smallest numbers of states.

This technique actually solves the problem of the construction of the partial automaton B with smaller number of states than the original automaton A, and the partial automaton mapping ψ induced by the automaton B coincides with the partial automaton mapping φ induced by the automaton A on the domain of definition of the mapping φ. In this case the domain of definition of the mapping ψ, generally speaking, is larger than the domain of definition of the mapping φ.

We shall show how the described minimization technique operates, using as an example the partial Mealy automaton A given by the following switching and output tables:

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline x & 2 & - & 2 & 4 \\ y & - & 3 & 4 & 4 \end{array}; \qquad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline x & u & - & u & u \\ y & - & u & v & v \end{array}.$$

Minimizing the given automaton, we obtain two initial possibilities of combining into 1-classes, leading to the smallest number of classes,

$$a_1 = (1, 3, 4.),\ b_1 = (2) \text{ and } a_1 = (1.2),\ b_1 = (3.4).$$

Let us consider the first possibility: the 1-table is written

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline x & b_1 & - & b_1 & a_1 \\ y & - & a_1 & a_1 & a_1 \end{array}.$$

From the 1-table we see that the class a_1 must be divided into two 2-classes: $a_2 = (1, 3)$ and $c_2 = (4)$; the third 2-class b_2 coincides with the 1-class $b_1 = (2)$; the 2-table will have the form

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline x & b_2 & - & b_2 & c_2 \\ y & - & a_2 & c_2 & c_2 \end{array}.$$

From the 2-table we see that the 3-classes coincide with the 2-classes, which are, consequently, the desired ∞-classes. In this variant the canonical minimization is represented by the table

$$\begin{array}{c|ccc} & a_2 & b_2 & c_2 \\ \hline x & b_2 & - & c_2 \\ y & c_2 & a_2 & c_2 \end{array}; \qquad \begin{array}{c|ccc} & a_2 & b_2 & c_2 \\ \hline x & u & - & u \\ y & v & u & v \end{array}.$$

In the second variant of the minimization, the partition into 1-classes $a_1=(1, 2)$ and $b_1=(3, 4)$ leads to the 1-table

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline x & a_1 & - & a_1 & b_1 \\ y & - & b_1 & b_1 & b_1 \end{array}$$

and to the partition into 2-classes $a_2 = (1, 2)$; $b_2 = (3)$; $c_2 = (4)$;. The 2-table is written

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline x & a_2 & - & a_2 & c_2 \\ y & - & b_2 & c_2 & c_2 \end{array},$$

which implies that the obtained partition into 2-classes will not break down further and, consequently, coincides with the partition into ∞-classes. The canonical minimization in this variant will be given by the tables

$$\begin{array}{c|ccc} & a_2 & b_2 & c_2 \\ \hline x & a_2 & a_2 & c_2 \\ y & b_2 & c_2 & c_2 \end{array}; \qquad \begin{array}{c|ccc} & a_2 & b_2 & c_2 \\ \hline x & u & u & u \\ y & u & v & v \end{array}.$$

The canonical minimizations obtained are essentially different: the first reacts to the input word $\underline{y}$ with the output word $\underline{v}$, while the second reacts with the output word $\underline{u}$.

§6. STRUCTURAL SYNTHESIS OF FINITE AUTOMATA

In the structural synthesis stage we select the elementary automata from which the synthesis of the structural diagram of the given abstract Mealy or Moore automaton is accomplished. We shall assume that the elementary automata are of two kinds: elementary automata with memory, or memory elements (triggers, delay lines, etc.), and elementary automata without memeory, also termed logic elements. For simplicity we shall limit ourselves to the case when we have available only one type of memory element.

The input and output signals of both the elementary automata and of the automaton under consideration as a whole are designated (coded) with a finite sequence of letters of some fixed finite alphabet, termed the structural alphabet. Usually, as the structural alphabet we choose the binary alphabet, consisting of two letters: 0 and 1. A second alphabet which plays a very important role in the structural synthesis stage is the alphabet consisting of the symbols of the internal states of the selected memory elements. We term it the state alphabet. The state alphabet may not coincide with the structural alphabet, however, in practice the binary alphabet is usually also chosen as the state alphabet.

One of the primary problems which is solved in the process if the structural synthesis of automata is the writing out of the so-called canonical equations which establish the relationship of the signals applied to the inputs of the memory elements to the output signals of these elements and to the signals applied to the input of the entire automaton as a whole. In order to ensure proper functioning of the circuit, we cannot permit direct participation of input signals, applied to the input of the memory elements, in the formation of the output signals which through the feedback circuits would be applied at that same instant of time to these inputs. In other words, the memory elements must be Moore automata and not Mealy automata. However, the complex automaton formed by these elements can, or course, be either a Moore or Mealy automaton.

Let us assume that the elementary automata with memory used in the structural synthesis are Moore automata. After making a corresponding shift of the reference of the time intervals for the output signals, we shall consider that the output signal at any instant of time _t_ of every memory element is determined by the internal state of th is element at _that same instant_ of time.

In order to have the possibility of synthesizing arbitrary automata with minimal consumption of memory elements, it is advisable to select as such elements Moore automata having a _complete system of transitions_ and a _complete system of outputs_, which for brevity we shall term _complete automata_. The completeness of the system of transitions means that for any pair of states of the automata there is an input signal which transforms the first element of this pair into the second. This requirement is equivalent to the requirement: in every column of the switching table there must be found all states of the given automaton. The completeness of the system of output in the case of the Moore automaton means that to each state of the automaton there is placed in correspondence its special output signal, differing from the output signals of the other states. Therefore, for the complete Moore automata it is natural to simply identify the output signals with the corresponding internal states of the automaton. We shall adhere to this method in the future.

Let us choose as a memory element some complete Moore automaton B. The internal states of this automaton are denoted by z_1, z_2, ..., ..., z_r ($r \geq 2$). According to the assumed condition they will also be the output signals. For the designation of the input signals of the memory element we shall use the letters s_1, s_2, ..., s_q. The alphabet

$(z_1, z_2, \ldots, z_r)$ is nothing other than the state alphabet. We shall select the structural alphabet somewhat later, and for the moment we shall show how to find the so-called canonical equation of the automaton whose memory is composed from the elements of the selected type.

Assume that we are given the abstract finite Mealy or Moore automaton A with the input alphabet $\underset{\cdot}{X} = (x_1, x_2, \ldots, x_n)$, the output alphabet $\underset{\cdot}{Y} = (y_1, y_2, \ldots, y_m)$ and the set of internal states $\underset{\cdot}{A} = (a_1, a_2, \ldots, a_p)$, specified by the switching function $\delta(a, x)$ and the output function $\lambda(a, x)$. Let the selected memory element B be given by the switching function $v(z, s)$. We pose the problem of finding the canonical equations of the automaton A under the condition that its memory is constructed from several copies $B^{(1)}, B^{(2)}, \ldots, B^{(k)}$ of the elementary automaton B.

For the construction of the automaton A the number k of memory elements must satisfy the condition $r^k \geq p$. In this case the various internal states a_i can be identified with the various sets of states of the automata $B^{(1)}, B^{(2)}, \ldots, B^{(k)}$. The process of such an identification will be termed the coding of the states of the automaton A in the chosen state alphabet. Of course, the coding process is in essence not unique. However, we shall not go into the details of the question associated with the selection of a particular coding variant, but shall consider that this variant is already given.

After coding, the states of the automaton A will be designated by the k-dimensional vectors $(z^{(1)}, z^{(2)}, \ldots, z^{(k)})$, whose components are the various letters $z_1\ z_2, \ldots, z_r$ of the state alphabet; the two-place output function $\lambda(a, x)$ of the automaton A is converted into the $(k + 1)$-place function $\lambda^0(z^{(1)}, \ldots z^{(k)}, x)$, which as before we shall term the output function. The equivalent of the switching function $\delta(a, x)$ after the coding will be the system of k $(k + 1)$-place functions $\varphi^{(1)}(z^{(1)},\dots, z^{(k)}, x). \dots, \varphi^{(k)}(z^{(1)}. \dots, z^{(k)}, x)$ of the transitions in the elementary memories. The function $\varphi^{(i)}$ determines the state into which the i-th memory element must transfer at the instant of time t + 1, if at the instant of time t the automaton A were in the state $(z^{(1)}, z^{(2)},\dots, z^{(k)})$, and to its input there was applied the signal x(i - 1, 2, ..., k; t = 0, 1, 2, ...).

The next important step is the construction of the excitation functions σ^{i} $(z^{(1)}, \dots, z^{(k)}, x)$ of the memory elements (i = 1, 2, ..., ..., k). The value of each such function $\sigma^{(i)}$ with the selected state $(z^{(1)}, z^{(2)}, \dots, z^{(k)})$ of the automaton A and the input signal x is determined as the input signal $S^{(i)}$ of the i-th memory element which causes the transfer in the i-th memory element due to the i-th switching function, i.e., the transfer $z^{(i)} \rightarrow \varphi^{(i)}$ $(z^{(1)}, \dots, z^{(k)}, x)$ (i = 1, ..., k). The input signal $S^{(i)}$ can be selected by more than one method. Therefore the construction of the excitation functions of the memory element from their switching functions is not unique.

The selection of the best method of construction of the excitation functions is associated with the problems of the following stage – the stage of combination synthesis. However, for many types of memory elements (delay lines, triggers with complementing input, etc.) the corresponding transfer is performed uniquely. For several types of elements we can idicate hybrid combinatorial-computational techniques which permit a simpler, in comparison with the general technique described method of finding the excitation functions [23].

The excitation functions $\sigma^{(i)}$, equated to the input signals $S^{(i)}$, determined by them, then give the sought canonical equations for the feedbacks in the automaton A. However, these function have a form which is still not completely satisfactory: the states of automaton A are coded in the iniversal (for the given type of memory elements) state alphabet, which does not depend on the choice of the automaton A; for the designation of the input and output signals use is made of various alphabets, including those which depend on the selection of the automaton A. Therefore it is necessary to fix the structural alphabet B, determined usually by the coding actually selected for the input signals of the memory element, and to code with the finite sequences of the letters of this alphabet not only the input signals $\sigma^{(i)}$ of the memory elements, but also the input and output signals x and y of the entire automaton as a whole. This coding transforms the system of excitation functions $\sigma^{(i)}$ found above into the new system of functions

$$\sigma_j^{(i)}(z^{(1)},\dots,z^{(k)}u_1, u_2,\dots,u_g)(i=1,2,\dots,k;\ j=1,2,\dots,l).$$

where each of the functions $\sigma^{(i)}$ is the input signal (letter of the structural alphabet) which must be applied to the j-th input channel of the i-th memory element at that time when the automaton A is in the state $(z^{(1)}, z^{(2)}, \dots, z^{(k)})$, and to its input channel there are applied the signals (letters of the structural alphabet) $u_1, u_2, \dots,$ $\dots, u_g$.

Similarly, the output function $\lambda^0(z^{(1)}, \dots, z^{(k)}, x)$ found above is replaced by the system of functions $\lambda_j(z^{(1)}, z^{(2)}, \dots, z^{(k)}, u_1,$ $u_2, \dots, u_g)(j = 1, 2, \dots, h)$, where the function λ_j determines the output signal (letter of the structural alphabet) appearing on the j-th output channel of the automaton A at the time when the automaton A is in the state $(z^{(1)}, z^{(2)}, \dots, z^{(k)})$, and to its input channels there are applied the signals $u_1, u_2, \dots, u_g$.

We term the resulting function $\sigma_j^{(i)}$ and λ_j respectively the structural excitation functions and the structural output functions of the automaton A.

In the case (usually encountered in practice) when both the structural alphabet and the state alphabet are binary alphabets, the structural excitation functions and the structural output functions can be considered as ordinary boolean functions. The problem of the following stage (the stage of combinational synthesis) amounts to the actual construction of the found functions from the elementary logic functions, realized by the selected logic elements. The methods of solution of this problem were discussed in §4 of the preceding chapter.

As the memory elements in the majority of the modern digital automata, use is made of the complete Moore automata with two internal states. It is interseting to analyze the question of how many and which of the elementary automata satisfy these properties. Let us consider the case when the complete Moore automaton with the two states 0 and 1 has only two input signals – x and y. From the conditions of completeness it follows that in each column of the switching table of the automaton there must be found both states – 0 and 1. This limitation leads to the existence of 4 possible switching talbes in all

$$\begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 0 \\ y & 1 & 1 \end{array}; \quad \begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 1 \\ y & 1 & 0 \end{array}; \quad \begin{array}{c|cc} & 0 & 1 \\ \hline x & 1 & 1 \\ y & 0 & 0 \end{array}; \quad \begin{array}{c|cc} & 0 & 1 \\ \hline x & 1 & 0 \\ y & 0 & 1 \end{array}.$$

After transformation of the input signals, the third table coincides with the first, the second with the fourth. Thus, there are only two essentially different automata of the required type, given by the swithcing tables

$$\begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 0 \\ y & 1 & 1 \end{array} \quad \text{and} \quad \begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 1 \\ y & 1 & 0 \end{array}.$$

If we set $x = 0$, $y = 1$, then the first table gives the well known memory element termed the delay (by one cycle) element, and the second gives the equally familiar element termed the complementing trigger. We note that the conventional electromagnetic relay with closing contacts can be considered as a delay element.

With an increase of the number of input signals there appear new types of memory elements: trigger with separate inputs, given by the switching table

$$\begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 1 \\ y & 0 & 0 \\ z & 1 & 1 \end{array},$$

the mixed trigger, given by the table

$$\begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 1 \\ y & 0 & 0 \\ z & 1 & 1 \\ u & 1 & 0 \end{array},$$

and others.

With the existence of only two internal states it is not useful to increase the number of input signals to more than four, since with a larger number of input signals some of them will begin to duplicate one another (cause identical transitions in the automaton). Therefore, it is not difficult to compose a catalog of all the complete essentially different Moore automata with two states (we shall consider as essentially different those automata whose swithcing tables do not convert one into the other with redesignated input signals). In addition to the four listed types of automata, the catalog will contain three more automata given by the switching tables

$$\begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 1 \\ y & 0 & 0 \\ z & 1 & 0 \end{array}; \quad \begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 1 \\ y & 1 & 1 \\ z & 1 & 0 \end{array}; \quad \begin{array}{c|cc} & 0 & 1 \\ \hline x & 0 & 0 \\ y & 1 & 1 \\ z & 1 & 0 \end{array}.$$

Of course, each of the listed types of automata permits various modifications as the result of different coding of the input signals in the binary alphabet. Let us consider as an example the complete synthesis (abstract and structural to the determination of the canonical equations) of the Moore automaton A which is a sequential binary squarer. The automaton A operates as follows: to its input there is applied a two-digit binary whole number, place-by-place, lower places first. At the output of the automaton the square of this number must appear, also sequentially, beginning with the lower digits. In other words, the automaton A must realize the following partial mapping φ:

$$\begin{aligned} 0000 &\to 0000 \\ 1000 &\to 1000 \\ 0100 &\to 0010 \\ 1100 &\to 1001. \end{aligned}$$

It is not difficult to verify that the mapping φ, continued to the initial segment of the words, satisfies the condition of automaticity. Denoting the zero signal at the input by the letter $\underline{x}$, and at the output by the letter $\underline{u}$, and correspondingly the ones signal on the input by $\underline{y}$ and on the output by $\underline{v}$, we write this correspondence in the form

$$\begin{aligned} xxxx &\to uuuu \\ yxxx &\to vuuu \\ xyxx &\to uuvu \\ yyxx &\to vuuv. \end{aligned}$$

In the resulting correspondence the output signal $\underline{u}$ represents the event $R = x \vee xx \vee xxx \vee xxxx \vee yx \vee yxx \vee yxxx \vee xy \vee xyxx \vee yy \vee yyx$, and the output signal $\underline{v}$ represents the event $Q = xyx \vee y \vee yyxx$. Words which do not occur in the events R and Q are forbidden. By use of the forbidden words we can extend these events without danger of impairing in the synthesized automaton its reaction to the specified words.

Since the events R and Q are disjoint, on the basis of what has been said we can replace the event R by the complement Q' of the event Q. In this case the automaton can synthesize just the one single event Q; the event Q' is automatically represented by the set of all non-labeled states of this automaton. Keeping in mind that for the labeling of the states the symbols $\underline{u}$ and $\underline{v}$ will be used somehow or other, we denote the event Q by the letter $\underline{v}$ and its complement Q' by the letter $\underline{u}$.

The process of abstract synthesis leads to the following marking of the regular expression for the event Q = v:

$$v = {}_{0}\,(\,{}_{\boxed{0}}\; y \;{}_{1}\; \vee \;{}_{\boxed{0}}\; x \;{}_{\boxed{2}}\; y \;{}_{\boxed{3}}\; x \;{}_{4}\; \vee \;{}_{\boxed{0}}\; y \;{}_{\boxed{1}}\; y \;{}_{\boxed{5}}\; x \;{}_{\boxed{6}}\; x \;{}_{4}\;)\;{}_{1,4}.$$

Here the same basic indices are assigned to a pair of corresponding places (index 1) and to a pair of similar places (index 4). The labeled automaton switching table corresponding to the marking is written

$$\begin{array}{c|cccccccc} & u & v & u & u & v & u & u & u \\ \hline & 0 & 1 & 2 & 3 & 4 & 5 & 6 & * \\ \hline x & 2 & * & * & 4 & * & 6 & 4 & * \\ y & 1 & 5 & 3 & * & * & * & * & * \end{array}.$$

Since the initial state 0 represents only an empty word, its label can be considered undetermined.

After transformation of the state (*) we obtain the table:

$$\begin{array}{c|cccccccc} & - & v & u & u & v & u & u & u \\ \hline & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline x & 2 & 7 & 7 & 4 & 7 & 6 & 4 & 7 \\ y & 1 & 5 & 3 & 7 & 7 & 7 & 7 & 7 \end{array}.$$

The set of states of the Moore automaton given by the last table can be divided into two 0-classes: a_0 = (0, 2, 3, 5, 6, 7) and b_0 = = (1, 4). Let us construct the 0-table and from it determine the 1-classes:

$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline & a_0 & b_0 & a_0 & a_0 & b_0 & a_0 & a_0 & a_0 \\ \hline x & a_0 & a_0 & a_0 & b_0 & a_0 & a_0 & b_0 & a_0 \\ y & b_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 & a_0 \end{array};$$

$$a_1 = (0), \quad b_1 = (1,4), \quad c_1 = (2,5,7), \quad d_1 = (3,6).$$

We construct the 1-table and determine the 2-class:

$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline & a_1 & b_1 & c_1 & d_1 & b_1 & c_1 & d_1 & c_1 \\ \hline x & c_1 & c_1 & c_1 & b_1 & c_1 & d_1 & b_1 & c_1 \\ y & b_1 & c_1 & d_1 & c_1 & c_1 & c_1 & c_1 & c_1 \end{array}; \quad a_2 = (0),\ b_2 = (1,4),\ c_2 = (2),\ d_2 = (3,6);\ f_2 = (5),\ g_2 = (7).$$

The 2-table and the 3-classes will have the form

$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline & a_3 & b_3 & c_3 & d_3 & b_3 & f_3 & d_3 & g_3 \\ \hline x & c_3 & g_3 & g_3 & b_3 & g_3 & d_3 & b_3 & g_3 \\ y & b_3 & f_3 & d_3 & g_3 & g_3 & g_3 & g_3 & g_3 \end{array}; \quad \begin{array}{l} a_3 = (0),\ b_3 = (1),\ c_3 = (2),\ d_3 = (3,6); \\ e_3 = (4),\ f_3 = (5),\ g_3 = (7). \end{array}$$

Since the 3-classes coincide, as is easily verifed, with the 4-classes, they will also be ∞-classes, so that the automaton can be constructed by combination of states 3 and 6. After corresponding renumbering of the states, the labeled switching table of the desired Moore automaton A is written

$$\begin{array}{c|ccccccc|} & - & v & u & v & u & u & u \\ \hline & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline x & 3 & 7 & 7 & 7 & 6 & 4 & 7 \\ y & 2 & 5 & 6 & 7 & 7 & 7 & 7 \end{array}.$$

Going to the structural synthesis, we choose as the memory element delay line with the switching table

$$\begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & 1 & 1 \end{array}.$$

Since the synthesized automaton has 7 states, while the memory element has 2 states, we must select 3 memory elements ($2^3 \geq 7$). Let u us denote their internal states by the letters z_1, z_2, z_3 and the input signals by s_1, s_2, s_3 (all these quantities can take the values 0 and 1). We denote the states of the automaton A by the values of the vector (z_1, z_2, z_3). Let us take the so-called nautral system of coding of the states, in which each state is coded by writing its number in the binary notation system

$$1 = 001;\ 2 = 010;\ 3 = 011;\ 4 = 100;\ 5 = 101;\ 6 = 110;\ 7 = 111.$$

Let us denote the physical input signal of the entire automaton A by the letter <u>c</u>, the physical output signal by <u>d</u>, and let us rewrite the labeled switching table of this automaton in accordance with the chosen coding system (such a table is usually called the <u>physical</u> switching table of the automaton in contrast to the previously considered <u>abstract</u> switching table in which no account was taken of the peculiarities of the coding).

d	—	1	0	1	0	0	0
$c \diagdown z$	001	010	011	100	101	110	111
0	011	111	111	111	110	100	111
1	010	101	110	111	111	111	111

The resulting physical switching table of the automaton gives the states $z_1' = z_1\ (t + 1)$, $z_2' = z_2(t + 1)$, $z_3' = z_3\ (t + 1)$ of the memory elements at each succeeding moment of time $t + 1$ as a boolean function of their states $z_1 = z_1\ (t)$, $z_2 = z_2(t)$, $z_3 = z_3\ (t)$ and of the input signal $c = c(t)$ of the automaton A at the present moment of time $\underline{t}$. from the switching table of the delay element we see that its state at any succeeding moment of time $t + 1$ coincides with the signal at its input at the present moment of time. Therefore we can consider that the written-out table gives the structural excitation functions of the sought automaton A

$$s_i = \sigma_i\ (z_1,\ z_2,\ z_3,\ c)\quad (i = 1,\ 2,\ 3).$$

We write out the tables of the values which determine respectively the functions s_1, s_2, s_3 immediately in the form of the Karnaugh map (see §4, Chapter 2)

Using the methods developed in the preceding chapter, we find the minimal disjunctive normal forms for the excitation functions (input signals of the memory elements) of the automaton A

$$s_1 = z_1 \vee z_2;\ s_2 = z_3 \vee \bar{z}_2 \vee \bar{z}_1\bar{c} \vee z_1c;\ s_3 = z_1c \vee z_1z_2z_3 \vee$$
$$\vee \bar{z}_2\bar{z}_3 \vee c\bar{z}_3 \vee \bar{z}_1\bar{c}.$$

We also determine the structural output function (which determines the output signal $\underline{d}$ of the automaton A as a function of the states of the elements of its memory) directly from the labeled physical switch-

z_1z_2 \ z_3c	00	01	11	10
00	—	—	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

z_1z_2 \ z_3c	00	01	11	10
00	—	—	1	1
01	1	0	1	1
11	0	1	1	1
10	1	1	1	1

z_1z_2 \ z_3c	00	01	11	10
00	—	—	0	1
01	1	1	0	1
11	0	1	1	1
10	1	1	1	0

ing table of the automaton A. The Karnaugh map for this function is written

z_1z_2 \ z_3'	0	1
00	—	—
01	1	0
11	0	0
10	1	0

From the Karnaugh map we easily find the minimal disjunctive form, which gives the required output function d,

$$d = \bar{z}_1\bar{z}_3 \vee z_2\bar{z}_4.$$

We note that all our functions were found to be determinate, not on all the sets, and we have extended their definitions in order to obtain representations for them which are as simple as possible.

Let us introduce as logic elements invertors, and also two-input AND and OR elements. Denoting them by circles with symbols of the operations corresponding to them $\neg$, $\wedge$, $\vee$, and denoting the memory (delay) elements by squares with the letters z_1, z_2 and z_3 inside, we obtain A which is shown in Fig. 9. This circuit includes, in addition to three

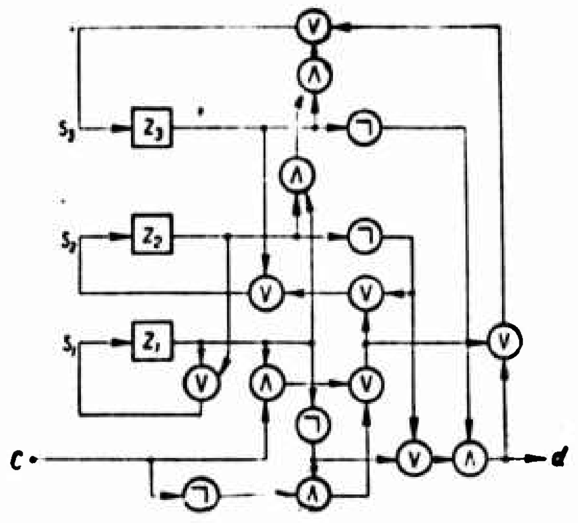

Fig. 9

delay elements, 4 invertors, 5 AND elements and 7 OR elements (16 logic elements and 3 delay elements in all).

Chapter 4

SELF-ORGANIZING SYSTEMS

§1. CONCEPT OF SELF-ALTERATION AND SELF-ORGANIZATION IN AUTOMATA

The concept of the algorithm and of the dicrete automaton is generally associated with the idea of their invariability with respect to time. Their corresponding alphabetic representations are pictured as something rigid, specified once and for all. With relation to the classical concept of the algorithm and to the usual understanding of the method of functioning of the discrete automaton this idea is to a certain degree justified. At the same time everyone knows that the most advanced of self-organizing systems are being simulated at the present time on the general purpose electronic computers, which are nothing other than discrete automata with a "rigid" structure, with the aid of programs, which are in essence algorithms written in some special form.

The contradiction arising here is to a considerable degree only apparent. The truth is that the difference between the "rigid" and the "self-altering" information converters is quite arbitrary in the majority of the cases and in determined not so much by the design of the converter itself, as by the organization of the experiment using the converter. The same information converter can in some conditions be considered to be rigid, unchanging, while under other conditions it can be considered as self-altering and self-organizing.

To make these statements more precise, let us consider any discrete automaton A with the input alphabet $\underset{\cdot}{X}$ the output alphabet $\underset{\cdot}{Y}$, the set of internal states $\dot{A}$ and the initial state a_0. The usual impression of the nature of the functioning of the automaton implicitly presumes that after the application to its input of some word $\underline{p}$ in the alphabet $\dot{X}$ and obtaining the corresponding output word $\underline{q} = \zeta(p)$ in the $\dot{Y}$ alphabet the automaton again returns to the initial state. Thus, at the moment of the beginning of the application of each new input word the automaton is always in the same state a_0. As a result of this the mapping induced by the automaton ζ is rigid, unalterable in the sense that the result of the conversion of any input word $\underline{p}$ by the mapping ζ depends only on the word $\underline{p}$ itself and not on the moment of time at which it was applied to the automaton input.

We will term each of the possible input words of the automaton a <u>question</u> and the corresponding output word a <u>response</u>. In this case "rigidity" of the automaton amounts to the fact that to a particular question it always and under all conditions gives the same response. The automaton is thereby deprived of any capability for learning and improvement of its responses.

However, the transition of the automaton into the initial state described above prior to each new question is not at all mandatory. Moreover, it is not specified directly in the definition of the functioning of the automaton which was given in §6 of Chapter 3. It is natural to define the functioning of the automaton so that the beginning of each succeeding question finds the automaton in the state in which it was after the termination of the answer to the preceding question. With this definition, the automaton which we previously considered to be rigid, unalterable will, generally speaking, change its responses in the course of time and can, in particular, be self-learning, self-improving, etc.

Let us now define more precisely the method of functioning of the discrete automaton with application to the theory of self-organizing systems. Extremely important concepts defining the method of functioning of the automaton are the concepts of the cycle and information cycling.

Assume that to the automaton input there is applied some (finite or infinite) sequence of letters: $x_{i_1} x_{i_2} \ldots x_{i_n}$. To it there corresponds some sequence of letters $y_{j_1} y_{j_2} \ldots y_{j_n}$ at the automaton output. Let us assume that we have identified some increasing sequence k, l, l, ... of moments of discrete time $(1 < k < \ell < \ldots)$. Then each pair of words $(x_{i_1} s_{i_2} \ldots x_{i_k},\ y_{j_1} y_{j_2} \ldots y_{j_k}),\ (x_{i_{k+1}} x_{i_{k+2}} \ldots x_{i_{\ell'}},\ y_{j_{k+1}} y_{j_{k+2}} \ldots y_{j_{\ell'}} \ldots$ will be termed a cycle, and the operation itself of the identification of the cycles will be termed the information (input and output) cycling operation for the automaton in question.

In the future we will assume that for each automaton under consideration there is identified a particular class of admissible sequences of input letters and that each such sequence (together with the corresponding sequence of output letters) is partitioned into cycles. The cycling operation is thus defined, generally speaking, not on some one pair of sequences, but on all pairs admissible sequences.

In the abstract approach to the concept of the cycle and cycling there is no additional meaining involved other than what has already been defined. However, in practice cycling always presumes that the pair of words (input and output) composing each such abstract cycle is in some sense a complete real cycle of functioning of the automaton, which can be considered separately from the remaining cycles. There are two cases which are encountered most frequently: the case when the first word of the pair (input word) is a question posed to the automaton, and the second word of the pair is the response to this question, and the case when the second word of the pair is, as before, the response, but the first, in addition to the question, includes in itself an evaluation of the given response as well. Of course, in both cases it is assumed that empty letters which may occur in either the first or second words need not be taken into consideration.

The situation which arises in these two cases is shown in Figs. 10 and 11 respectively.

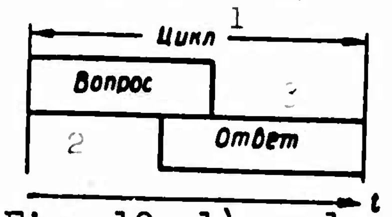

Fig. 10. 1) cycle; 2) question; 3) response.

We note that in the general case it is frequently advisable in the design of automata to provide for partial (and sometimes even complete) overlapping of the response and question (begin the response before the question is terminated). This situation is reflected in Figs. 10 and 11. In performing the cycling operation the boundaries of the cycles are also

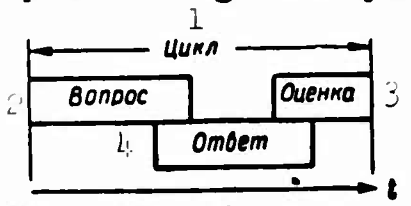

Fig. 11. 1) cycle; 2) question; 3) response; 4) valuation.

determined basically by two methods. We can, first, simply fix some natural number k and require that the input and output word in each cycle contain exactly k letters (including empty letters as well). We will term this k-cycling. Second, we can define the boundaries of the cycles by fixing for this purpose a special letter or word, termed a label. For the separation of the cycles it is most convenient to place such a label at the beginning of each successive question (here we will consider that the label is a part of the question). We will agree to call this method of cycling label cycling. It is obvious that the combination of letters fixed as labels must be used exclusively for this purpose. A label can also be used within a cycle (for example, for indicating the beginning of an evaluation), but this label must be different from the label which indicates the boundary of the cycle. In the design of the automaton it is frequently convenient to provide for the automaton to put out a special label at the end of each response. We will consider that in the operation of the automaton there are encountered only admissible sequences of input signals, and that for each such sequence the corresponding partition into cycles has been performed.

The ordered sequence of cycles preceding the given cycle C in a particular fixed admissible sequence of the automaton A is termed the learning history of the automaton A for the cycle C.

It is natural to term an automaton self-improving or self-learning if in the course of the lengthening of the learning history it improves its responses. This definition, of course, in no way lays pretense to exactness and must be considered to be preliminary. The definitions for the concept of the improvement (self-learning) of the automata will be made more precise in one of the following sections after a preliminary consideration of the probability-theoretic concepts which are necessary to such definitions. However, it is useful to mention here the directions of this further definition. First of all it is necessary to refine the concept of the quality of the response with the aid of the introduction of some numerical evaluation of the response. Under this condition we can put exact meaning into the concept of improvement of the quality of the response which was used above in the definition of the self-improvement of the automata.

Further, we must keep in mind that even the automata with the most clearly marked tendency to self-improvement do not necessarily improve their responses absolutely to all questions. Here we must consider the improvement of the quality of the resposnes on the average. The same is true of the learning history. Some relatively in frequently encountered learning histories can obviously lead to deterioration of the average quality of the responses, however if the remaining learning history leads to a sharp improvement of the response quality the automaton as a whole can be considered self-improving.

Finally, it is obvious that we must differentiate self-improvement which is prespecified ahead of time by the automaton designer (regardless of the form of the learning history) and the really self-triggered self-improvement which is determined by the learning history which actually takes place and which therefore is not planned ahead of time. It is clear that only the second type of self-improvement is actually deserving of this name. As for the first type, in this case the designer actually places the correct responses in the automaton ahead of time, but in order to simulate the process of self-improvement he forces the automaton keep this information under judgement for a certain time. As a result the automaton at first gives responses of poor quality and only at the end of some period (some number of cycles) does it begin, using the information which has been stored in it, to give correct answers. However, we can hardly term this sort of improvement of the quality of the automaton responses with time self-improvement.

All that we have said gives an idea of the difficulties which must be overcome in the exact definition of the concept of self-improvement. In a similar situation is the concept of <u>self-organization</u>, which it seems to us is somewhat more general than the concept of self-improvement. With self-improvement there must of necessity be improvement of the quality of the responses. With self-organization the quality of the responses may not be determined at all. It is only necessary that in the course of learning the automaton on the average increases the definiteness of these responses. The corresponding refinement of the definition will be given after the introduction of the necessary probability-theoretic concepts.

In the refinement of the concepts of self-organization and self-improvement it is convenient to make use of the so-called <u>cyclic reduction</u> of the automata. Cyclic reduction is defined for the automata in which the set of admissible input sequences is fixed and the cycling of the input and output information has been performed. With satisfaction of these conditions, for any automaton A the input and output alphabets can be replaced as follows: the letters of the new input alphabet $\dot{X}'$ are considered to be all the different input words of all cycles in all the admissible sequences, the letters of the new output alphabet $\dot{Y}'$ are similarly considered to be all the different output words of the indicated cycles.

For any state $\underline{a}$ of the automaton A and any letter x' of the alphabet $\dot{X}'$ (input word of some cycle), we use $\delta'(a, x')$ to denote the state into which the automaton transitions from the state $\underline{a}$ under the action of the input word x'. We use $\lambda'(a, x')$ to denote the output word delivered by the automaton A under the action of the input word x' in the case when the state $\underline{a}$ is taken as the initial state. Any admissible input sequence of the automaton A can be considered ad the sequence $x'(1)x'(2) \ldots$ of letters of the new input alphabet $\dot{X}'$. Let us consider the set $\dot{A}'$ of all those states of the automaton A into which it can be switched from the initial state a_0 by the input words of the form $x'(1)\, x'(2) \ldots x'(k)$ $(k \geq 0)$, i.e., by all possible initial segments of the various admissible input sequences. The initial state a_0 itself of necessity occurs in this set.

Now it is not difficult to construct the automaton A' in which the set of internal states is the set $\dot{A}'$, the input alphabet coincides with the set $\dot{X}'$ and the output alphabet coincides with the set $\dot{Y}'$. The switching and output functions of this automaton will be the functions δ' and λ' defined above, and the initial state will be the state a_0.

It is assumed that only admissible input sequences (rewritten in the alphabet X') will be applied to the input of the constructed automaton. In this case, as is easily verified, the definition of the automaton A is completely correct: there are sufficient states and output letters to completely describe the functioning of the automaton under the influence of any admissible input sequence.

We agree to term the automaton A' thus constructed the cyclic reduction of the original automaton A. Obviously the information cycling will be a 1-cycling in the automaton A'. In other words, in the cyclic reduction of any automaton both the questions and the responses are single-lettered.

With cyclic reduction of automata the number of their internal states can only diminish or, at the least, remain unchanged. The number of letters of the input and output alphabets will, generally speaking, increase. It is clear that with k-cycling of the original information cyclic reduction cannot cause a transition from finite alphabets to infinite. However, in the case of label cycling such a transition is completely possible – after cyclic reduction a finite input or output alphabet may be transformed into an infinite one.

Let us consider as an example the cyclic reduction of the automaton A with the three states 1, 2, 3, the two input letters x, y and the two output letters u, v whose switching and output functions are given by the respective tables

$$\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline x & 2 & 2 & 2 \\ y & 3 & 2 & 1 \end{array}; \qquad \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline x & u & v & u \\ y & v & u & v \end{array}.$$

Assuming all the input sequences admissible and taking the state 1 as the initial state, as a result of the cyclic reduction we arrive at the automaton A' with two states, four input letters x_1 = xx, x_2 = = xy, x_3 = yx, x_4 = yy, four output letters $v_1 = uu,\ v_2 = uv,\ v_3 = vu,\ v_4 = vv.$

The switching and output functions of this automaton are given by the respective tables

$$\begin{array}{c|cc} & 1 & 2 \\ \hline x_1 & 2 & 2 \\ x_2 & 2 & 2 \\ x_3 & 2 & 2 \\ x_4 & 1 & 2 \end{array}; \qquad \begin{array}{c|cc} & 1 & 2 \\ \hline x_1 & v_3 & v_4 \\ x_2 & v_1 & v_3 \\ x_3 & v_3 & v_2 \\ x_4 & v_4 & v_1 \end{array}$$

With the use of the cyclic reduction a very graphic solution is found of the question of whether the automaton being considered is rigid or self-altering with respect to the given cyclization. Actually, we formulate the following proposition.

In order that the discrete automaton A with given cyclization be rigid (i.e., it does not alter its responses to the same question in the course of time) it is necessary and sufficient that after cyclic reduction the output function $\lambda'(a, x)$ not depend on the states of the reduced automaton A'.

Independence of the output function on the states of the automaton means, obviously, equivalence between all the elements of each row of the output table of the automaton (elements standing in different rows can, of course, be different).

With application to the example considered above, the proposition just formulated immediately discloses the self-variability of the automaton A for the case of 2-cycling of its input information.

It is easy to see that the automaton in which the output function does not depend on the states can be replaced by its equivalent (i.e., inducing the same alphabetic mapping) automaton having one single internal state. The automaton with a single internal state is in essence an automaton without memory. In the abstract theory of automata it is shown that every discrete automaton A can be minimized, i.e., in other words, can be replaced by its equivalent automaton B (absolute minimization of the automaton A) having the smallest number of states among all the automata which induce the same alphabetic mapping as does automaton A.

If after cyclic reduction of any discrete automaton A (with given cyclization of the information) we then perform an absolute minimization, we obtain an operation which we shall term <u>complete cyclic reduction</u> of the considered automaton A (for the given cyclization). The validity of the following proposition resutls from the above considerations.

In order that the discrete automaton A with given information cycling have the property of time independence of its elements, it is necessary and sufficient that as the result of complete cyclic reduction of the automaton A we obtain an automaton without memory.

The converse is also true: the existence of a nontrivial memory in the automaton obtained as the result of complete cyclic reduction of the considered automaton A means that the automaton A is (relative to the given cyclization) self-adaptive.

The relativity of the property of self-adaptability of automata (its dependence on the method of cycling the input information) is easily illustrated by the example of the automaton C with two states, given by the switching and output tables

$$\begin{array}{c|cc} & 1 & 2 \\ \hline x & 2 & 1 \\ y & 1 & 2 \end{array};\quad \begin{array}{c|cc} & 1 & 2 \\ \hline x & u & v \\ y & v & u \end{array}.$$

With any admissible input sequences in the case of 2-cycling of the input information, the result of cyclic reduction of the automaton C will be an automaton with a single state. Thus, even without minimization we obtain an automaton without memory and, in view of the criterion formulated above, we arrive at the conclusion on the rigidity, invariability of the automaton C. At the same time, with 1-cycling of the input information the automaton C must be, obviously, considered to be self-adaptive. In practical problems we can quite easily differentiate the rigid and self-adaptive automata simply because the cyclization of the input information is prespecified.

We note that in the criteria considered and in the examples discussed on the basis of these criteria we spoke not of self organization or self-improvement, but only of self-adaptivity of the automata. The analysis of examples of self-organization and self-improvement will be made in the later sections after creation of the corresponding mathematical basis and the introduction of precise definitions.

In the remainder of the present section we shall consider one terminological question. That is the usage of the terms "system" or "automaton" in combination with the concepts of self-adaptation, self-organization, self-improvement and self-learning. As we see from the discussions already presented, all these concepts can be developed for the discrete automata. However, with this approach to the matter we essentially lose the possibility of penetrating into the structure of the corresponding process (self-adaptation, self-organization, etc.).

The study of the structure of the self-adaptation and self-organization processes is facilitated with the representation of such processes not in the form of individual automata (algorithms), but in the form of systems of automata (algorithms). In the simplest case such a system consists of two automata (algorithms). The first of these, t termed the <u>operational</u> automaton (algorithm), directly processes the information applied to the system input. The second automaton (algorithm), termed the <u>controlling</u> or <u>learning</u> automaton (algorithm), evaluates the results of the functioning of the operational automaton (algorithm) and introduces into it the suitable changes (in the case when we are considering automata, these changes are introduced into the switching and output functions of the operational automaton).

Over the controlling automaton (algorithm) of the first stage there can be placed the controlling automaton (algorithm) of the second stage, whose function is to evaluate the operation of the automaton (algorithm) of the first stage and introduce into it the required changes. By analogy we can introduce controlling automata (algorithms) for the third, fourth, and any higher stages. We shall term the hierarchy of automata (algorithms) which arise in this fashion systems and shall develop the concepts of self-adaptation, self-organization and self-improvement for them.

Of course, in the abstract sense, any, no matter how complex, system of automata is equivalent to a single automaton, however such reduction of the systems to individual automata leads to loss of the possibility of study of certain properties of such systems which are of practical interest, primarily the laws for the circulation of information within the system itself. Therefore, in the future we shall deal not only with automata (algorithms) considered abstractly but also with systems of automata (algorithms) for whose study the internal structure is of particular interest, i.e., the relations between the individual automata (algorithms) composing the system.

§2. SOME AUXILIARY INFORMATION FROM PROBABILITY THEORY

In the present section we will present certain information from probability theory which is needed for out further constructions. In view of the fact that this presentation is of an auxiliary nature, proofs of most of the propositions formulated will not be included. If needed, the reader can find the corresponding proofs in the monographs of Feller [81] and Kramer [48]. It is assumed that the reader is familiar with such elementary concepts of the theory of probability as the concept of the event, event probility, etc.

The concept of the random quantity is of very essential importance in the construction of the theory of the self-organizing systems. We shall limit ourselves to the consideration of only the random quantities which take on real values. Here, in addition to the conventional so-called univariate random quantities, we consider also the multivariate random quantities whose values will be the finite ordered ensembles of real numbers or, what is the same, the real vectors of a particular (finite) dimension.

It is also important to differentiate continuous and discrete random quantities. The continuous random quantity can take any values in a particular region (open set) of the corresponding vector space, for example on some interval of the real axis (including the entire axis as well) in the case of the univariate random quantities. However, the totality of the possible values of the discrete random quantity can be only the discrete sets of points, i.e., those sets, each point of which can be inclosed in a sphere (possibly of very small radius) which does not contain other points of the same space. An example of the discrete set might be the set of all points of some Euclidean space which have integral coordinates.

The property of randomness of the quantities we have considered manifests itself in the so-called trials. In each trial the considered random quantity takes a particular value from the domain of its definition. The probability that the random quantity will take a particular value is determined by the distribution law of this random quantity. The distribution law of the discrete random quantity $\underline{x}$ (univariate or multivariate) is specified with the aid of the real function $f(x)$ defined for all values which the given random quantity can take so that for any value x_1 the magnitude of $f(x_1)$ is equal to the probability that the random quantity $\underline{x}$ will take the value x_1 in the given trial.

The domain of definition of the discrete random quantities which we are considering can be either finite or denumerably infinite. It is evident that in both cases the normalization condition

$$\sum_i f(x_i) = 1, \tag{47}$$

is satisfied, where the summation is assumed to extend over the entire region of definition of the given random quantity.

When the random quantity x is continuous, its distribution law is given with the aid of the so-called probability density function $\varphi(x)$. This function is presumed defined in the region M of definition of the considered random quantity x and integrable in this region. With each successive trial the probability p(N) that the random quantity x will take a value from some subregion N of its region of definition is equal to the integral of the probability density taken with respect to this subregion:

$$p(N) = \int_N \varphi(x)\,dx. \tag{48}$$

Whence follows directly the satisfaction of the normalization condition

$$\int_M \varphi(x)\,dx = 1. \tag{49}$$

Two random quantities x and y are termed mutually independent if when the quantity x takes a particular value there is no change of the distribution law of the quantity y and vice versa. Similarly the independence of any set of random quantities implies that when all the quantities occurring in this set, other than the quantity x, take any values there is no change of the distribution law of this latter quantity with any choice of the quantity x from the indicated set.

Trials performed with a particular random quantity x are termed independent trials if the distrubution law of the quantity x remains unchanged in each trial and, consequently, does not depend on the values which the quantity $\underline{x}$ took in the previous trials.

The domain of definition of the continuous random quantity can always, if need be, be extended over the entire space, assuming that everywhere except in the original domain of definition the probability density is equal to zero.

We can also approximate the continuous distribution laws with the discrete laws and vice versa. In the first case it is sufficient to partition the domain of definition M of the corresponding continuous random function $\underline{x}$ into a finite number of sufficiently small (not only in volume, but also in diameter) subdomains $M_1, M_2, \ldots, M_n$, select within each such subdomain M_i a point x_i and introduce the discrete distribution law f(x) on the selected points, setting $f(x_i) = \int_{M_i} \varphi(x)\,dx$ $(i = 1, 2, \ldots, n)$, where $\varphi(x)$ is the probability density of the original continuous random quantity. In this case the probabilities previously associated with the corresponding subdomains are concentrated in the individual points.

With the reverse transition from the discrete distribution law to the continuous, on the contrary, there is a "diffusion" of the probability initially concentrated in the individual points x_i into the corresponding subdomains M_i so that for the probability density function $\varphi(x)$ thus appearing the following relations are valid

$$\int_{M_i} \varphi(x)\,dx = f(x_i) \qquad (i = 1, 2, \ldots, n).$$

Frequently it is necessary to consider the infinite sequences of discrete distribution laws $f_i(x)$ $(i = 1, 2, \ldots)$, having some continuous distribution law with a probability density function $\varphi(x)$ in the form of its so-called limit distribution law. The following precise meaning is embedded in the concept of the limit distribution. First, the domains of the values M_i of the discrete random quantities with the distributions laws $f_i(x)$ converge to the domain of the values M of the continuous random quantity $\underline{x}$. In the cases we consider this convergence will mean that all M_i are contained in M, and for any arbitrarily small positive number ε, any arbitrarily large number N, and for any point $\underline{x}$ from M in each of the sets M_i with $i > N$ there is at least one point removed by less than ε from the point $\underline{x}$. Second, for any subdomain P of the domain M and for any arbitrarily small positive number δ for all numbers $\underline{i}$, beginning with some number, the following inequality must be satisfied

$$\left|\int_P \varphi(x)dx - \sum_{x \in P \cap M_i} f_i(x)\right| < \delta. \qquad (50)$$

The summation in the left side of this formula is taken over all the points from M_i contained in the subdomain P.

The concepts of the <u>mean value</u> (mathematical expectation) of the random quantity and its <u>second order central moments</u> are of great importance for the further constructions.

Let $\vec{x} = (x_1, x_2, \ldots, x_n)$ be an n-dimensional continuous random quantity with the probability density function $\varphi(x_1, x_2, \ldots, x_n)$. As noted above, without losing generality the function φ can be considered determinate over the entire infinite space. Then the mean value of the random quantity $\underline{x}$ is defined as the vector $\vec{m} = (m_1, m_2, \ldots, m_n)$ computed from the equation

$$\vec{m} = (m_1, m_2, \ldots, m_n) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \vec{x}\varphi(x_1, x_2, \ldots, x_n)\, dx_1\, dx_2 \ldots dx_n. \qquad (51)$$

The second order central moments λ_{ik} are determined by the equations

$$\lambda_{ik} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} (x_i - m_i)(x_k - m_k)\varphi(x_1, x_2, \ldots, x_n)\, dx_1 dx_2 \ldots dx_n \quad (i, k = 1, 2, \ldots, n). \qquad (52)$$

For the univariate random quantity $\underline{x}$ there is the natural second order central moment determined by the equation

$$d = \int_{-\infty}^{\infty} (x-m)^2 \varphi(x)\,dx. \qquad (53)$$

This moment is usually termed the variance of the corresponding distribution.

It is natural to transfer all these concepts and their definitions to the discrete random quantities as well.

To do this we need only in equations (51)-(53) replace the integration by summation extended over the entire domain of definition M of the corresponding discrete (vector) quantity $\vec{x}$, and in place of the probability density function $\varphi(x_1, x_2, \ldots, x_n)$ write the probability distribution function of this quantity $f(\vec{x})$. As a result, equation (51), for example, is rewritten

$$\vec{m} = \sum_{\vec{x} \in M} \vec{x} f(\vec{x}). \qquad (54)$$

All the remaining equations are changed similarly.

For the multivariate random quantities it is convenient to combine the second order central moments λ_{ik} into the matrix $\|\lambda_{ik}\|$ and construct from them (in the case when they are finite) $Q(t_1, t_2, \ldots, t_n) = \sum_{i,k} \lambda_{ik} t_i t_k$. With the aid of orthogonal conversion (rotation) of the coordinate system this form can always be reduced to the form $\sum_{i=1}^{m} \alpha_i (t_i')^2$, where t_i' are new coordinates and α_i are positive coefficients ($i = 1, 2, \ldots, m$). Forms of this type are termed positive semi-definite. If $m = n$, i.e., if the number of squares after reduction of the form Q is exactly equal to the dimension of the space, then the form Q is termed <u>positive definite</u>. In this case its determinant $|Q| = |\lambda_{ik}|$ is of necessity nonzero and (strictly) positive.

For the positive definite form $Q = \sum_{i,k} \lambda_{ik} t_i t_k$ we can define the inverse form $Q^{-1} = Q^{-1}(t_1, t_2, \dots, t_n) = \sum_{i,k} \mu_{ik}t_it_k$, whose coefficients are given by the equations $\mu_{ik} = Q_{ik}/|Q|$, where Q_{ik} is the algebraic complement of the element λ_{ik} of the determinant $|Q| = |\lambda_{ik}|$ $(i, k = 1, 2, \dots, n)$. This form will again be positive definite. The matrix obtained $\|\mu_{ik}\|$ will obviously be the inverse of the matrix $\|\lambda_{ik}\|$, since the latter is symmetrical (of course, the matrix $\|\mu_{ik}\|$).

The following fundamental result [48] is of great importance in probability theory.

Theorem 1. If the n-variate random (continuous or discrete) quantities $x_1, x_2, \dots, x_k$ are independent and have the same distribution with finite second order central moments λ_{ik} for which the form $Q(t_1, t_2, \dots, t_n) = \sum_{i,k} \lambda_{ik}t_it_k$ is positive definite, and with mean value equal to zero, then as $k \to \infty$ the quantity $x = \frac{1}{\sqrt{k}}(x_1 + x_2 + \dots + x_k)$ has a limit continuous distribution also with zero mean value and with the (univariate) probability density function

$$\varphi(t_1, t_2, \dots, t_n) = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{|Q|}} e^{-\frac{1}{2}Q^{-1}(t_1, t_2, \dots, t_n)}.$$

In the case when the form Q is degenerate, we turn to the consideration of some subspace L of the original space, replacing the random quantities $x_1, x_2, \dots, x_k$ by their projections on the subspace. The subspace L is chosen so that the new form Q of the central moments obtained as the result of the indicated projection is already positive definite, while the projection onto any subspace perpendicular to it would lead to a degenerate form (such a space always exists). The application of theorem 1 in the constructed space L gives in this case a distribution in the original space as well, since all possible values of the random quantities $x_1, x_2, \dots, x_k$ (and this means their sums as well) lie in this subspace with the probability 1.

In several cases we limit ourselves to the selection of the subspace K of maximal possible dimension of the number of all subspaces with the nondegenerate form Q. Let us show that the subspace L can be obtained from the subspace K as the result of a nondegenerate linear transformation.

If in the conditions of theorem 1 the mean value of the quantities $x_1, x_2, \ldots, x_k$ is nonzero, then, denoting this mean value by $m = (m_1, m_2, \ldots, m_k)$, we find easily that the mean value of the random quantity $x = 1/\sqrt{k}\,(x_1 + x_2 + \ldots + x_k)$ will be the quantity $(m_1\sqrt{k}\; m_2\sqrt{k}, \ldots, \ldots, m_n\sqrt{k})$ and that with sufficiently large k a good approximation for the distribution law of the random quantity x will be the continuous law with a univariate probability density function of the form

$$\varphi(t_1, t_2, \ldots, t_n) = \frac{1}{(2\pi)^{\frac{n}{2}}\sqrt{|Q|}}\, e^{-\frac{1}{2}Q^{-1}(t_1 - m_1\sqrt{k},\, t_2 - m_2\sqrt{k},\, \ldots,\, t_n - m_n\sqrt{k})}. \qquad (55)$$

Let us now apply theorem 1 and equation (55) to the so-called binomial distribution. The binomial distribution arises as the result of the conduct of independent trials using the so-called Bernoulli scheme. This scheme, in addition to the property of independence of the trials, is also characterized by the fact that with each trial only two outcomes are possible, occurring with the probabilities p and $q = 1 - p$ respectively. Let us term the first outcome success and ther second failure of the trial and let us introduce the random quantity x_i, taking the value 1 in case of success of the i-th trial and the 0 in case of its failure ($i = 1, 2, \ldots, n$).

The random quantity $k = x_1 + x_2 + \ldots + x_n$ is clearly equal to the total number of successes with n independent trials. Let us denote by C_n^k the number of combinations of n with respect to k (by definition $C_n^0 = 1$), then it is easy to find that the distribution (discrete) law of the random quantity k is given by the function

$$f_n(k) = C_n^k p^k q^{n-k} \quad (k = 0, 1, \ldots, n). \tag{56}$$

This law is termed the binomial distribution law since the quantity $C_n^k p^k q^{n-k}$ is obviously the $(n - k + 1)$th term of the expansion of the expression $(p + q)^n$ using the Newton binomial formula.

The random quantities $x_1, x_2, \ldots, x_n$ have the same distribution law with the mean value $m = 1 \cdot p + 0 \cdot (1 - p) = p$ and the variance (second central moment) $d = (1 - p)^2 \cdot p + (0 - p)^2(1 - p) = p(1 - p)$. From theorem 1 and equation (55) it follows that for sufficiently large n a good approximation for the distribution law of the quantity $x = k/\sqrt{n} = 1/\sqrt{n}(x_1 + x_2 + \ldots + x_n)$ will be the distribution law with the probability density function of the form

$$\varphi(x) = \frac{1}{\sqrt{2\pi p(1-p)}} e^{-\frac{(x-p\sqrt{n})^2}{2p(1-p)}}. \tag{57}$$

The distribution law with the probability density function of the form $\frac{1}{\sqrt{2\pi a}} e^{-\frac{(x-m)^2}{2a}}$ (with $a > 0$) we shall term the (generalized) univariate normal distribution law. It is not difficult to see that the value of the random quantity distributed according to this law is equal to m and that its variance is equal to a.

It is easy to see that with multiplication of the normally distributed random quantity x by the constant factor c the new quantity $y = cx$ will also be a normally distributed random quantity and its mean value will be c times larger and the variance c^2 times larger in comparison respectively with the mean value and the variance of the original quantity x.

Comparing the results obtained with equation (57), we come to the following proposition.

Theorem 2. With a sufficiently large number n of Bernoulli trials with probability of success p the distribution law for the total number of successes k can be approximately expressed by the normal law with the probability density function of the form $\varphi(k)=\frac{1}{\sqrt{2\pi np(1-p)}}e^{-\frac{(k-pn)^2}{2np(1-p)}}$. The mean value of the random quantity corresponding to this law is equal to pn and its variance is equal to np(1 – p), which agree with the exact values of the mean value and the variance of the original discrete random quantity k.

As the result of the fact that in the derivation of the statement contained in theorem 2 the quantity x was multiplied by the factor $\sqrt{n}$, the continuous distribution φ for the quantity k obtained in theorem 2 does not possess, generally speaking, the property of the limit distribution for the original (discrete) distribution f of the quantity k with unbounded increase of the number of trials n.

However, it is not difficult to note that with sufficiently large values of n the probabilities calculated in accordance with the distributions φ and f of finding the quantity k in any intergral whose length is of the order of the quantity $\sqrt{n}$ (i.e., has the form $c\sqrt{n}$, where c is a constant) will differ from one another by arbitrarily small amounts.

In practice we usually need to calculate the probability of finding the quantity k on intervals of the form [pn, pn ± $z\sigma$], where the quantity $\sigma = \sqrt{np(1-p)}$, equal to the square root of the variance of the distribution φ (and this means of the distribution f as well), is termed the mean square variation (or the mean square error) of the distributions φ and f. The following theorem is valid.

Theorem 3. For any positive number z the probability $\rho(z)$ that the total number of successes in n Bernoulli trials with a probability of success p will be found in the interval [pn, pn ± $z\sqrt{np(1-p)}$], is expressed by the equation $\rho(z) \approx \Phi(z) = 1/\sqrt{2\pi} \times \int_0^z e^{-\frac{x^2}{2}}dx$. With any z, by choosing n sufficiently large, we can make the error in the calculation of $\rho(z)$ using this equation arbitrarily small.

We show the numerical values of the function $\Phi(z)$ for some values of x with four decimal places: $\Phi(1) = 0.3413$; $\Phi(2) = 0.4772$; $\Phi(3) =$ $= 0.4986$; for $z \geq 4$ the values of $\Phi(z)$ differ from 0.5000 by less than half a unit of the fourth decimal place.

We term the approximate equation in the condition of theorem 3 the de Moivre-Laplace formula. As indicated in theorem 3, the accuracy of this formula increases with the increase of the number n of trials performed.

Let us consider a series of independent trials, each of which has m different outcomes, and let p_i $(p_i > 0)$ denote the probability of the i-th outcome of the trial $(i = 1, 2, \ldots m)$. We denote the total number of trials conducted by the letter n and the number of those which terminated with the i-th outcome – $k_i (i = 1, 2, \ldots, m)$. It is easy to see of the quantities k_i, considered by itself, is distributed in accordance with the binomial law. We pose the problem of finding the joint (multivariate) distribution law of several quantities k_i, for example the quantities $k_1, k_2, \ldots, k_r$ $(1 \leq r \leq m)$. It is not difficult to verify that the solution of this problem is given by the equation

$$f(k_1, k_2, \ldots, k_r) = \\ = \frac{n!}{k_1!\,k_2!\ldots k_r!(n-k_1-k_2-\ldots-k_r)!} \times \\ \times p_1^{k_1} p_2^{k_2} \ldots p_r^{k_r} (1 - p_1 - p_2 - \ldots - p_r)^{n-k_1-k_2-\ldots-k_r}. \tag{58}$$

For this distribution law, which is customarily termed the polynomial distribution law, we can obtain a continuous approximation just as we did above for its particular (univariate) case. To do this let us consider the multivariate random quantities $x^j = (x_1^j, x_2^j, \ldots, x_r^j)$, such that the quantity x^j takes one of the values (100 ... 0), (010 ... ... 0), ..., (00 ... 01) or (000 ... 00) in accordance with the outcome of the j-th trial of the series we consider – first, second, ..., r-th or any different from the $(j = 1, 2, \ldots, n)$. All the random quantities x^j have the same distribution law, their mean values are equal, obviously, to $(p_1, p_2, \ldots, p_r)$. The second central moment λ_{ii} is equal, obviously, to the quantity $(1 - p_i)^2 p_i + (0 - p_i)^2(1 - p_i) = p_i(1 - p_i)$ $(i = 1, 2, \ldots, r)$. The second central moment λ_{ik} with $i \neq k$ also is easily calculated: $\lambda_{ik} = (1 - p_i)(0 - p_k)p_i + (0 - p_i)(1 - p_k)p_k + (0 - p_i)(0 - p_k)(1 - p_i - p_k) = - p_ip_k$.

The determinant $|\lambda_{ik}|$ of the matrix $\|\lambda_{ik}\|$ of the central moments will be equal in this case, as is easily shown, to the product $p_1p_2 \ldots p_r(1 - p_1 - p_2 - \ldots - p_r)$. Thus, the matrix $\|\lambda_{ik}\|$ will be degenerate only in the case when $r = m$. In all the remaining cases the quadratic form $Q(t_1, t_2, \ldots, t_r) = \sum_{i,k}\lambda_{ik} t_it_k = \sum_{i=1}^{r} p_i (1 - p_i)t_i^2 - \sum_{i \neq k} p_ip_kt_it_k$ will be positive definite, since its determinant $|Q| = |\lambda_{ik}|$ is positive.

Applying theorem 1 to the random quantity $y = 1/\sqrt{n}(y_1 + y_2 + \ldots + y_n)$, where $y_i = (x_1^i - p_1, x_2^i - p_2, \ldots, x_r^i - p_r)$, we come to the conclusion that with $r < m$ it has a limit (as $n \to \infty$) distribution law with a probability density function of the form

$$\frac{1}{(2\pi)^{\frac{r}{2}}\sqrt{|Q|}} e^{-\frac{1}{2}Q^{-1}(t_1, t_2, \ldots, t_r)}.$$

The multivariate random quantity $z = (k_1/n - p_1, k_2/n - p_2, \ldots, k_r/n - p_r)$ is connected with the quantity $\underline{y}$ by the relation $z = 1/\sqrt{n} = y$ and will therefore have the same distribution law, but with a variance n-fold less than the variance of the quantity $\underline{y}$. Consequently, the probability density function of the distribution law for $\underline{z}$ is written

$$\frac{1}{\left(\frac{2\pi}{n}\right)^{\frac{r}{2}}\sqrt{|Q|}} e^{-\frac{n}{2}Q^{-1}(z_1, z_2, \ldots, z_r)}.$$

It is now not difficult to establish the following result.

Theorem 4. Let there be given the series of independent trials with $\underline{m}$ different outcomes, having resprctively the probabilities

$p_1, p_2, \ldots, p_m$ (same for all trials). If in the series of n trials we use $k_1, k_2, \ldots, k_m$ to denote the number of trials terminating respectively by the 1st, 2nd, ..., m-th outcome, then for any a priori given positive number ε for the probability ρ of the simultaneous satisfaction of all the inequalities $|k_i/n - p_i| < \varepsilon$ ($i = 1, 2, \ldots, m$) there exists the estimate $\varrho > 1 - \frac{a}{n^{\frac{m-1}{2}}} e^{-bn}$ (where a and b are positive constants not dependent on n).

For the proof of this theorem we note, first, that all the inequalities $\left|\frac{k_i}{n} - p_i\right| < \varepsilon$ ($i = 1, 2, \ldots, m$) are obviously satisfied if there are satisfied the m − 1 inequalities

$$\left|\frac{k_i}{n} - p_i\right| < \frac{\varepsilon}{m-1} \quad (i = 1, 2, \ldots, m-1). \tag{59}$$

Actually

$$\left|\frac{k_m}{n} - p_m\right| = \left|\frac{n - k_1 - k_2 - \ldots - k_{m-1}}{n} - (1 - p_1 - p_2 - \ldots - p_{m-1})\right| = \left|\left(\frac{k_1}{n} - p_1\right) + \left(\frac{k_2}{n} - p_2\right) + \ldots + \left(\frac{k_{m-1}}{n} - p_{m-1}\right)\right| < \frac{\varepsilon}{m-1}(m-1) = \varepsilon.$$

It follows from the considerations preceding the formulation of theorem 4 that the quantities $z_i = k_i/n - p_i$ ($i = 1, 2, \ldots, m - 1$) have a limit (as $n \to \infty$) distribution law with the probability density function of the form $\varphi(z_1, z_2, \ldots, z_{m-1}) = \alpha n^{\frac{m-1}{2}} e^{-nP(z_1, z_2, \ldots, z_{m-1})}$, where α is a positive constant and P is a positive definite quadratic form with coefficients not depending on n For sufficiently large n the probability β that at least one of the inequalities (59) is not satisfied has, obviously, an upper estimate of the form

$$\beta < \iint\limits_{R_1} \ldots \int \varphi(z_1, z_2, \ldots, z_{m-1})\, dz_1 dz_2 \ldots dz_{m-1}, \tag{60}$$

where the region R_1 is the outer portion of the hypercube bounded by the hyperplanes $z_i = \delta_i$ ($\delta_i < \varepsilon/m - 1$, $i = 1, 2, \ldots, m - 1$).

After rotation of the coordinate axes for the purpose of reducing the form P to the sum of squares with positive coefficients b_1, b_2, ..., b_{m-1} we can in the hypercube turned relative to the new axes inscribe the new hypercube R, bounded by the hyperplanes $z_i' = \delta$ ($\delta < \delta_1$, $i = 1, 2, \ldots, m-1$). Integration of the transformed probability den- sity function over the region external to the new hypercube gives again an estimate of the form (60), which can be strengthened by replacing all the coefficientd b_1, b_2, ..., b_{m-1} by the smallest coefficient among them, designated by g.

As a result we obtain the new estimate

$$\beta < an^{\frac{m-1}{2}}\left(2\int_{\delta}^{\infty} e^{-gnx^2}dx\right)^{m-1}. \qquad (61)$$

Since

$$\int_{\delta}^{\infty} e^{-gnx^2}dx < \frac{1}{\delta}\int_{\delta}^{\infty} e^{-gnx^2}x\,dx = \frac{1}{\delta}\cdot\frac{1}{2gn}e^{-gn\delta^2},$$

it is easy to obtain the final estimate

$$\beta < \frac{a}{(g\delta)^{m-1}n^{\frac{m-1}{2}}}e^{-gn\delta^2}. \qquad (62)$$

Denoting $\alpha/(g\delta)^{m-1}$ by the letter a and $g\delta^2$ by the letter b, we obtain the required estimate, for the present, it is true, for all n beginning with some possibly quite large value. We can, however, also take account in the derived estimate of the remaining finite set M of values of n by increasing, in case of necessity, the constant a. Since the probability ρ is clearly greater than zero, it is sufficient to select the quantity a larger enough so that the right side of the estimate under discussion becomes negative for all values of n belonging to the set M.

Thereby theeorem 4 is fully proved.

In concluding the present section we shall describe still another frequently encountered distribution — the so-called Poisson distribution. This distribution can be treated as an approximation for the distribution with the condition than the number of trials n is large, the probability of success ρ in each trial is small, and the product $\lambda = np$ is not small, but also is not large. In this case the probability α that exactly k trials lead to success is expressed by the approximate equation

$$\alpha \approx e^{-\lambda}\frac{\lambda^k}{k!}. \tag{63}$$

In particular, for $k=0$ $\alpha \approx e^{-\lambda}$.

The Poisson distribution has a maximum of the probability with a maximal value of k satisfying the inequality $k \leq \lambda$. In the theory of discrete self-organizing systems we encounter the Poisson distribution in the organization of teaching automata words or sequences of words of differing length. With a random selection of the words being used in the teaching, the Poisson distribution frequently gives a sufficiently good approximation for the distribution law of the word lengths.

We note that the mean value of a quantity having a Poisson distribution (63) is equal to λ.

§3. A QUANTITATIVE MEASURE OF SELF-ORGANIZATION AND SELF-IMPROVEMENT IN AUTOMATA

In the first section we encountered the concept of self-adaptation in automata: it is natural to term automaton self-adaptive if it changes in the course of time its responses to the questions fed to it (for some cycling of the input and output information). However, not every self-adaptation should be identified with self-organization. On the basis of the intuitive idea of self-organization, we should term self-organizing that automaton which improves the organization of its possible learning histories. For the quantitative characteristic of this improvement it is natural to make use of the probabilistic-theoretic concept known as entropy.

We shall use the entropy concept only for the discrete random quantities. Let there be given the discrete random quantity x with the domain of definition R and with the distribution law f(x). In this case the entropy of this quantity, or, what is the same, the entropy of the distribution f(x), is the term given to the negative sum, taken over the region of definition R of the given random function, of the products of the probabilities f(x) and their logarithms

$$H = -\sum_{P} f(x) \log f(x). \qquad (64)$$

In the use of this equation it is assumed that for f(x) = 0 the product f(x) log f(x) is zero. Any positive number, strictly greater than unity, can be selected as the base of the system of logarithms. It is easy to see that with a change of the base of the logarithm system the values of the entropies for all the distribution laws are multiplied by the same constant factor. In practice, use is commonly made either of the binary (with base two), natural, or decimal logarithms.

As is shown in information theory (see, for example, Goldman [32]), entropy is the natural measure of the indefiniteness of the values of the random quantity: the greater this indefiniteness, the larger the value of the entropy. In particular, if the random quantity can take only two values with the probabilities p and q = 1 – p respectively, the maximal value of the entropy is achieved with equality of these probabilities: p = q = 1/2, which corresponds to the intuitive concept on the maximal possible indefiniteness in this case. If, however, one of the probabilities p or q vanishes, then, as is easily seen, the value of the entropy also vanishes, which again is in good agreement with common sense, since in this case there is actually no indefiniteness.

With combination of the two independent random quantities x and y into one multivariate (with dimension equal to the sum of the dimensions of the quantities x and y) random quantity z = (x, y), the entropy of the distribution of the quantity z is equal to the sum of the entropies of the distributions of the quantities x and y.

Actually, if the distribution laws of the quantities x and y are given by the functions $f_1(x)$ and $f_2(x)$, then the distribution law of the quantity z is obviously given by the product of these functions $f_1(x)f_2(y)$. The entropy of the quantity z (H_z) is then calculated from the equation

$$H_z = -\sum_{x\in P_1}\sum_{y\in P_2} f_1(x)f_2(y)\log[f_1(x)f_2(y)] =$$
$$= -\sum_{y\in P_2} f_2(y)\sum_{x\in P_1} f_1(x)\log f_1(x) - \sum_{x\in P_1} f_1(x)\sum_{y\in P_2} f_2(y)\log f_2(y) = H_x + H_y,$$

where P_1 and P_2 are the regions of definition of the quantities x and y, and H_x and H_y are their entropies.

The property of the entropies of the independent distributions which leads to the formation of their sum when these distributions are combined into one is termed the entropy additivity property.

For the automata operating using the simple question-response cycle (without evaluation of the quality of the response), we can approach the definition of the extent of the self-organization with the aid of the consideration of two entropy characteristics – the learning entropy and the examination entropy of the automaton. In the following discussion we shall follow basically the work [25].

Let $(p_1, p_2, \ldots, p_n) = P$ be the sequence of questions (input words) supplied to the automaton in its learning period. We will term this sequence the learning sequence. Let us assume that in a particular fixed series of experiments with the automaton, for each learning sequence P there is given the probability $\rho(P)$ of the appearance of this sequence in the experiments of the series under consideration (it is assumed that within the limits of the given series this probability does not vary from experiment to experiment). This specifies some distribution R of the probabilities $\rho(P)$ of the learning sequences. The entropy of this distribution, which we shall term the <u>learning entropy</u> with the given learning distribution law, is calculated from the familiar equation

$$H^R\ (\text{learn}) = -\sum_P \varrho(P) \log \varrho(P). \qquad (65)$$

For definiteness we agree to use natural logarithms for the computation of the entropies.

In the case when the automaton A and its initial state a_0 are fixed, every distribution of the probabilities $\rho(P)$ of the learning sequences uniquely determines some distribution of the probabilities $\alpha(a)$ on the set of all states of this automaton. Here $\alpha(a)$ denotes the probability that after termination of the automaton learning process it will be in the state <u>a</u>. If we use S_a to denote an event at the input of the automaton, representable by the state <u>a</u> (set of input words transferring the automaton from the initial state into the state <u>a</u>), then we obtain

$$\alpha(a) = \sum_{P \in S_a} \varrho(P). \qquad (66)$$

where the summation extends to all words of the form $P = p_1\ p_2\ \ldots\ p_n$, contained in S_a (for brevity of writing, the sequence of words $P = {}$ $=(p_1,\ p_2,\ \ldots,\ p_n)$ is identified here with the word $p_1\ p_2\ \ldots\ p_n$, composed from the elements of this sequence).

Now let us fix some probability distribution $\gamma(p)$ of the questions <u>p</u> applied to the automaton after termination of its learning process. The distributions $\alpha(a)$ and $\gamma(p)$ together with the switching and output functions of the considered automaton A uniquely define the probability distribution $\beta(p, q)$ for the pair "question (p) – answer (q)". We term the entropy of the latter expression the examination entropy and denote it by H^Q(exam). We use Q to denote the so-called law of experimentation with the automaton, which is the combination of the two distribution laws — the distribution of the learning sequences and the distribution of the examination questions.

The quantity H^Q(exam) is determined from the equation

$$H^Q\ (\text{exam}) = -\sum \beta(p, q) \log \beta(p, q). \qquad (67)$$

Using if necessary the operation of cyclic reduction of the automata, we can, without losing generality, consider only sequences of single-letter questions and responses. Here the learning sequences P are converted into words consisting of the individual components of their question-letters arranged in the order in which they were applied to the automaton in the learning process.

For the further construction of the theory it is necessary to specify some class of laws of experimentation with the automaton and assign to each law Q occurring in this class some probability, or, in the case of the continuous distribution laws, some probability density $\varphi(Q)$.

The simplest case is the scheme of independent trials, when at each step, both in the learning regime and in the examination regime, the probability $\gamma(P)$ of the appearance of any given question is constant and depends only on this question. In view of the limitation to only 1-cycled automata, the specification of the law Q for experimentation with the automaton is equivalent in this case to the assignment of certain probabilities $\nu_i = \nu(x_i)$ of the appearance at the input of the automaton of each of the letters x_i of its input alphabet. The sum of all the ν_i, of course, must be equal to unity in this case.

In the case of the scheme of independent trials the law of experimentation with the automaton is naturally identified with the vector $\nu = (\nu_1, \nu_2, \ldots)$, consisting of the probabilities of the appearance of the different input letters (it is assumed that the input alphabet is ordered in some fashion). The class of laws is naturally identified with the set of all vectors $\nu = (\nu_1, \nu_2, \ldots)$, satisfying the natural limitations $0 \leq \nu_i \leq 1$ and $\Sigma\nu_i = 1$ $(i = 1, 2, \ldots)$, with a uniform distribution law given on this set. We agree to call the scheme of independent trials with this selection of class of distribution law the uniform scheme of independent trials. Here we limit ourselves to the case when the length of the learning sequence is fixed, or in case of necessity we shall assume that these lengths are described by some distribution law (most frequently Poissonian).

If there is given some law of experimentation Q, then it, as noted above, includes in itself two distribution laws — the law of distribution of the learning sequences and the law of distribution of the examination questions. The corresponding random quantities are to be considered independent in the case of the usual organization of the experiments on the self-improving automata. Therefore the entropy of the joint distribution of these two quantities, which we shall agree to term the <u>entropy of the corresponding law of experimentation</u> Q and designate by H^Q, will be equal to the sum of two entropies — the learning entropy H^Q(learn) and the entropy of the examination questions H^Q(quest). The latter entropy must not be confused with the examination entropy H^Q(exam) which relates not to the distribution of the examination questions, but to the distribution of the question-response pairs. The examination entropy depends not only on the distribution of the questions and the distribution of the learning sequences, but also on the automaton itself, while the entropy of the law of experimentation with the automaton does not depend on the automaton.

Let us assume that in the class K of laws of experimentation with the automaton there is fixed some law Q_0 which has the maximal possible entropy H^{Q_0}. Introducing increments of the entropies of experiment and examination by the equations

$$\Delta H^Q = H^Q - H^{Q_0}(\text{exam}) = H^Q\ (\text{exam}) - H^{Q_0}(\text{exam}).$$

we obtain the possibility for any automaton A and class K of laws of experimentation with the automaton (with the probability density $\varphi(Q)$) to introduce the two averaged characteristics

$$s(A, K) = -\int_K \Delta H^Q\ (\text{exam})\ \varphi(Q)\, dQ; \tag{69}$$

$$z(A, K) = \int_K \frac{\Delta H^Q(\text{exam})}{\Delta H^Q}\ \varphi(Q)\, dQ. \tag{70}$$

The integrals in these equations are taken over the region consisting of all the laws of the considered class K. The larger the value of these integrals, the greater the average capacity of the considered automaton A for self-organization. The zero value corresponds to the absence of capability for self-organization, and negative values indicate that with improvement of the organization of the learning, the organization of the responses of the automaton on the average deteriorates. In other words, the automaton behaves as a "self-disorganizing" system rather than as a "self-organizing" system.

Since equation (70) leads to considerably more complex computations than equation (69), we shall select as the basic quantitative criterion for the evaluation of the capability of an automaton for self-organization the criterion $\underline{s}$ (A, K) rather than the criteria $\underline{z}$ (A, K).

Let us consider as an example the two automata A and B whose switching and output functions are given by the tables:

for automaton A $\begin{array}{c|cc} & 1 & 2 \\ \hline x & 1 & 1 \\ y & 2 & 2 \end{array}$; $\begin{array}{c|cc} & 1 & 2 \\ \hline x & u & v \\ y & u & v \end{array}$;

for automaton B $\begin{array}{c|cc} & 1 & 2 \\ \hline x & 1 & 2 \\ y & 2 & 2 \end{array}$; $\begin{array}{c|cc} & 1 & 2 \\ \hline x & u & v \\ y & u & v \end{array}$.

In these tables the numbers 1 and 2 denote the states of the automata, the letters x, y denote the input signals (questions), and letters u, v denote the output signals (responses).

As the class K of distribution laws we select that class in which the probabilities of the occurrence of the examination questions x and y are equal, and the distribution laws of the learning sequences result from the scheme of independent trials with the probabilities of the independent trials with the probabilities of the occurrence of the questions x and y equal to p and 1 – p respectively (p runs through all the values from 0 to 1 in the limits of the class K with equal probabilities). In addition, we fix the length n of the learning sequences, and we denote the criterion s(A, K) corresponding to the selected value of n by $s_n(A, K)$.

The automaton A will be in the state 1 if the last question given to it during learning was x, and in the state 2 if the last question given it was y. This implies that the probabilities of the question-response pairs will be equal: for the pair (x, u) – 1/2 p, for the pair (y,u)–1/2 p, for the pair (x, v) – 1/2 (1 – p) and for the pair (y, v) – also 1/2 (1 – p). Consequently, the examination entropy

$$H^0\ (\text{exam}) = -\frac{1}{2}p\ln\frac{1}{2}p - \frac{1}{2}p\ln\frac{1}{2}p - \frac{1}{2}(1-p)\ln\frac{1}{2}(1-p) - $$
$$-\frac{1}{2}(1-p)\ln\frac{1}{2}(1-p) = -p\ln p - (1-p)\ln(1-p) - \ln\frac{1}{2}.$$

The maximal entropy of the experimentation will obviously be with $p = 1 - p = 1/2$. In this case the examination entropy is determined by the expression

$$H^{Q}\,(\text{exam}) = -\frac{1}{2}\ln\frac{1}{2} - \frac{1}{2}\ln\frac{1}{2} - \ln\frac{1}{2} = -2\ln\frac{1}{2}.$$

The increment of the examination entropy

$$\Delta H^{Q}(\text{exam}) = -p\ln p - (1-p)\ln(1-p) + \ln\frac{1}{2}.$$

The probability density of the laws Q in the selected class is clearly equal to unity. Application of equation (69) gives

$$s_n(A, K) = \int_0^1 \left(p\ln p + (1-p)\ln(1-p) - \ln\frac{1}{2}\right)dp = \ln 2 - \\ -\frac{1}{2} \approx 0.19.$$

The automaton B will be in the state 1 only when the learning sequence consists of only x's. The probability of this is obviously p^n. Hence the probabilities of the examination pairs (x, u) and (y, u) are equal to $1/2\ p^n$, and the probabilities of the pairs (x, v) and (y, v) are equal to $1/2\ (1 - p^n)$. Just as in the case of finding $s_n(A, K)$, we find ΔH^Q (exam), as a result of which we obtain the sequence of equations

$$s_n(B, K) = \int_0^1 \left[p^n \ln\frac{1}{2}p^n + (1-p^n)\ln\frac{1}{2}(1-p^n) - \frac{1}{2^n}\ln\frac{1}{2^{n+1}} - \right.$$
$$\left. -\left(1-\frac{1}{2^n}\right)\ln\frac{1}{2}\left(1-\frac{1}{2^n}\right)\right]dp = \frac{n\ln 2}{2^n} - \frac{n}{(n+1)^2} - \left(1-\frac{1}{2^n}\right)\times$$
$$\times\ln\left(1-\frac{1}{2^n}\right) + \int_0^1 (1-p^n)\ln(1-p^n)\,dp = \frac{n\ln 2+1}{2^n} - \frac{n}{(n+1)^2} -$$
$$-\frac{1}{n}\left[\frac{1}{1\left(1+\frac{1}{n}\right)\left(2+\frac{1}{n}\right)} + \frac{1}{2\left(2+\frac{1}{n}\right)\left(3+\frac{1}{n}\right)} + \dots\right] -$$
$$-\left(\frac{1}{1\cdot 2\cdot 2^{2n}} + \frac{1}{2\cdot 3\cdot 2^{3n}} + \dots\right).$$

Using this last relation we obtain the estimate

$$s_n(B, K) < \frac{n \ln 2 + 1}{2^n} - \frac{n}{(n+1)^2} + \frac{1}{4n}.$$

From this estimate we easily learn that with n $\geq$ 5 the quantity s_n (B, K) is negative. In other words, with learning using sequences of length greater than 4, in the selected class of laws of experimentation the automaton B is on the average "self-disorganizing", while the automaton A under the same conditions discloses capability for self-organization.

We note that the conclusion on the capability or the incapability of the automton for self-organization depends on the selection of the class of laws of experimentation with this automaton. If, for example, in the example considered we select as K the class of laws of experimentation which results from the uniform scheme of independent trials, then as is easily verified, the automaton B would also become self-organizing on the average, although the magnitude of this self-organization would remain less than that of the automaton A.

With transition from the concept of self-organization to the concept of self-learning we can no longer be satisfied with the purely probabilistic-theoretic concepts. It is necessary to introduce the concepts which characterize the particular directionality of the self-organization process. To do this it is most natural to introduce the real function f(p, q) defined on the set of all possible question (q) – response (q) pairs, whose value characterizes the quality of any response to any given question p.

As we noted above, for any given automaton A with fixed initial state a_0 the specification of the law Q of the probability distribution $\rho(P)$ on the learning sequences P uniquely determines the probability distribution $\alpha(a)$ on the set of the automaton states. Let us further denote by $q = \lambda(a, p)$ the response of the automaton A, which has been first reduced to the state a, to the question p. The quantity $f^Q = \sum_{p,a} f(p, \lambda(a, p)) \gamma(p) \alpha(a)$ is the averaged criterion of the quality of the responses of the automaton to the "examination" when it has been taught by the sequences distributed according to the law Q (γ (p) is the probability of the appearance of the question p in the examination).

It is natural to use the term self-learning content of the automaton A to denote the difference $f^Q - f^{Q_0}$ where Q_0 is the a priori probability distribution law of the learning sequences, known to the designer at the time of construction of the automaton, and Q is the a posteriori distribution law which actually exists for some class of learning experiments. As a rule, the entropy of the distribution Q_0 is greater than the entropy of the distribution Q.

If now there is given the class K of a posteriori distribution laws Q, with the probability density $\varphi(Q)$, then the integral b (A, K) = $= \int_{Q \in K} (f^Q - f^{Q_0}) \varphi(Q) dQ$ is the averaged quantitiative characteristic for the capability of the considered automaton for self-learning (for the selected class K, the automaton A and the real function f).

§4. AUTOMATA WITH RANDOM TRANSITIONS

In addition to the determinate automata, in the theory of self-organizing systems we must consider automata which have random transitions. As is known (see Chapter 3), in the determinate automaton the specification of the preceding state a(t − 1) and the current input signal x(t) uniquely determines the next following state a(t) into which the automaton transfers under the influence of this input signal from the state a(t − 1). In the automaton with random transitions the specification of the pair a(t − 1), x(t) determines only the probability $p_{ij}(x)$ of the transition of the automaton from the state a(t − 1), which we denote by a_i, into any other state a_j under the influence of the input signal x(t) = x.

It is easy to see that the determinate automaton can be considered as a particular case of the automaton with random transitions in which for each $\underline{x}$ with any given $\underline{i}$ only precisely one of the probabilities $p_{ij}(x)$ is equal to unity, and all the remaining probabilities are equal to zero.

It is natural to specify every automaton with random transitions with the aid of the system of matrices $\|p_{ij}(x)\|$, where $\underline{x}$ runs sequentially through all the input signals of the automaton. Of course, in addition to such matrices there must also be given the output functions and the initial state of the automaton.

The matrices $\|p_{ij}(x)\|$ have the property that the sum of the elements of any of their rows is equal to unity. We shall assume also that there are no states in the automaton for which the probabilities of the transition from all the other states are equal to zero. This means, obviously, that the matrices $\|p_{ij}(x)\|$ do not have columns composed only of zeros. In addition, all the elements of each of the matrices $\|p_{ij}(x)\|$ are nonnegative real numbers which do not exceed unity.

Matrices satisfying the three listed properties are customarily termed stochastic matrices. Thus, in the case of automata with random transitions the role of the switching function is played by the function $\|p_{ij}(x)\|$, which uniquely maps the set of all input signals of the automaton into the set of stochastic matrices.

Of particular interset are the automata with random transitions which have one single (constant) input signal. Such automata are studied in classical probability theory under the name of uniform (discrete) Markov chains. The output signals in such automata are ignored (or identified with the states), which permits specifying these automata with the aid of a single stochastic matrix. For definiteness, it is customarily considered that the first row (and the first column as well) of this matrix corresponds to the initial state of the automaton (Markov chain).

The Markov chains also have another (non-automaton) interpretation – in the terms customarily used in probability theory. This interpretation is based on the concept of trials, considered in the preceding section. However, here we must consider not independent trials, but the trials in which the probabilities of particular outcomes of each successive trial depend on the outcome of the directly preceding trial and do not depend directly on the outcomes of all the remaining trials (the set itself of possible outcomes does not change from trial to trial). The there arises the matrix $\|p_{ij}\|$ of the so-called transition possibilities. Any element p_{ij} of this matrix is the probability of the jth outcome in each successive trial under the condition that the outcome of the trial directly preceding it was i.

It is easy to see that such treatment is completely equivalent to the automaton treatment: the trial outcome is, essentially, simply another name for the state of the automaton (having a single input signal) with random transitions. There is, it is true, one difference: in the automaton with random transitions there was fixed a completely determined initial state, in the Markov chains it is customary to specify the probabilities of the various outcomes of the initial trial $p_1, p_2, \ldots, p_n$, which corresponds to the random selection of the initial state of the automaton, so that the ith state can be selected as the initial state with the probability p_i $(i = 1, 2, \ldots, n)$.

Thus, for a more complete analogy with the Markov chains it is necessary to consider not the simple automata with random transitions (having a single input signal) but the so-called random automata in which not only the transition function but also the selection of the initial state is random, and if, in addition, the output signals are taken into consideration, then the output function must, generally speaking, be random. In other words, the output function must specify not simply the output signal, but some probability distribution on the set of all possible output signals.

The Markov chains (or, correspondingly, the random automata) are termed finite or infinite depending on whether the possible set of outcomes (or, correspondingly, the set of states of the automaton) is finite. We shall require for the random finite automata of general form also finiteness of the set of their input and output signals. We shall limit ourselves to the study of only the uniform Markov chains, i.e., those chains in which the matrix of the probability transition probabilities is constant. We will not encounter nonuniform Markov chains (with matrix of the transition probabilities which depends on time) in the future. Therefore for brevity we shall speak only of Markov chains, meaning each time, if not otherwise stipulated, that we mean uniform chains.

Let us consider the automaton A with random transitions (Markov chain) whose transition probability matrix is $P = \| p_{ij} \|$. As mentioned above, the arbitrary element p_{ij} of this matrix is the probability of the transition of the automaton A from the ith state into the jth. It is important to emphasize that here we are speaking of the transition in one cycle (i.e., the interval between two neighboring moments of discrete automaton time). It is easy to see that the product $p_{ik}p_{kj}$ is the probability of the transition of the automaton A from the ith state into the jth in two cycles with the condition that the automaton passes through the kth state.

The sum $\sum_k p_{ik}\, p_{kj}$, extended over all the states, obviously gives the total probability of the transition of the automaton A from the ith state into the jth in two cycles. Moreover this sum is clearly the element of the matrix $P \cdot P = P^2$ standing at the intersection of the ith row and the jth column. We obtain similarly: the probability of the transition of the automaton A from the ith state into the jth after three cycles of operation is equal to the (i, j)th element of the matrix $P^2 \cdot P = P^3$. Continuing similarly, we come to the following proposition.

Theorem 1. For there random automaton A with a single input signal (uniform Markov chain) whose transition probability matrix is P, the probability of transition from the ith state into the jth after exactly n cycles is equal to the element of the matrix P^n standing at the intersection of the ith row and the jth column $(n = 1, 2, 3, \ldots)$.

It is natural to term the elements of the matrix P^n the transition probabilities after n steps. For the determination of these probabilities in the case of the finite Markov chains we make use usually of the so-called Perron equation which is derived in matrix theory. Let us first recall certain definitions and concepts of this theory.

Let there be given the matrix $P = \|p_{ij}\|$ of nth order. The determinant

$$P(\lambda) = \begin{vmatrix} \lambda - p_{11} & -p_{12} & \ldots & -p_{1n} \\ -p_{21} & \lambda - p_{22} & \ldots & -p_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ -p_{n1} & -p_{n2} & \ldots & \lambda - p_{nn} \end{vmatrix}$$

is a polynomial of nth degree in λ, termed the characteristic polynomial of the matrix P. The roots of this polynomial are termed the eigenvalues of the matrix P.

Let us denote by E the unit matrix of nth order. Then the element of the matrix $\lambda E - P)^{-1}$, located at the intersection of the ith row and the jth column will be equal to $1/P(\lambda) \cdot P_{ji}(\lambda)$ where $P_{ji}(\lambda)$ is the algebraic complement of the element of the determinant $P(\lambda)$ located at the intersection of its jth row and ith column.

Now let the matrix $P = \|p_{ij}\|$ have the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_r$. We denote by m_i the multiplicity of the ith number λ_i, i.e., in other words, the maximal number s such that the characteristic polynomial $P(\lambda)$ is divided by $(\lambda - \lambda_i)^s$, and we define the polynomial $\psi_i(\lambda)$ by the equation

$$\psi_i(\lambda) = \frac{P(\lambda)}{(\lambda-\lambda_i)^{m_i}}.$$

Then the element $p_{ij}^{(k)}$ of the matrix $P^{(k)}$, located at the intersection of the ith row and the jth column can be determined from the equation

$$p_{ij}^{(k)} = \sum_{\nu=1}^{r} \frac{1}{(m_\nu - 1)!} D_\lambda^{m_\nu - 1} \left[\frac{\lambda^k P_{ji}(\lambda)}{\psi_\nu(\lambda)}\right]\Bigg|_{\lambda=\lambda_\nu}. \qquad (71)$$

Equation (71) is the Perron equation. In it $D_\lambda^{m_\nu - 1}$ denotes the derivative with respect to λ of order $m_\nu-1$. Substitution of the value $\lambda =$ $= \lambda_\nu$ must be performed after the differentiation. The derivation of the Perron equation can be found in any monograph on the theory of finite Markov chains (see, for example, Romanovskiy [68]).

The Perron equation takes a particularly simple form in the case when all the eigenvalues of the matrix P have a multiplicity equal to unity, i.e., when $m_1 = M_2 = \ldots = m_r = 1$. It is clear that in this case $r = n$. Since the factorial of zero is unity, and the derivative of zero order denotes the absence of any differentiation, then for this particular case we obtain the simple equation

$$p_{ij}^{(k)} = \sum_{\nu=1}^{n} \frac{\lambda_\nu^k P_{ji}(\lambda_\nu)}{\psi_\nu(\lambda_\nu)} \quad (i, j = 1, 2, \ldots, n). \qquad (72)$$

We shall term equation (71) the general, and equation (72) the special Perron equation. Equations (71) and (72) permit the solution of one very important problem of the theory of finite Markov chains — the problem of finding the so-called limit distribution. If there exists the limit $\lim_{k\to\infty} P^k = P^\infty$, then it is natural to term the elements of the matrix $P^{\infty} = \|p_{ij}^{\infty}\|$ the limit transition probabilities. Having the initial distribution, i.e., the probabilities $p_1, p_2, \ldots, p_n$ of the various outcomes of the initial trial, we can obtain the probabilities p_i^{∞} of the various states in the limit distribution from the equations

$$p_i^{\infty} = \sum_{j=1}^{n} p_j p_{ji}^{\infty} \quad (i = 1, 2, \ldots, n). \tag{73}$$

It appears natural to assume that after a sufficiently large number of transitions of the random automaton characterizing the Markov chain, the effect of the initial distribution of the state probabilities on the distribution of the states obtained as the result of these transitions can be made arbitrarily small. In other words, the limit distribution obtained using equation (73) must not depend on the initial distribution $(p_1, p_2, \ldots, p_n)$. If the limit distribution has this property, then the corresponding Markov chain is termed ergodic. The ergodicity property will obviously hold if and only if for any given i all the elements p_{ji}^{∞} $(j = 1, 2, \ldots, n)$ are identical, i.e., in other words, when all the rows of the matrix of the limit transition probabilities are identical.

It can be shown that the moduli of the eigenvalues of the stochastic matrices cannot exceed unity. It is also not difficult to see that the eigenvalue for any stochastic matrix is unity. If all the remaining (non-unity) eigenvalues of the stochastic matrix M are strictly less than unity in modulus, then the matrix M and the corresponding Markov chain are termed proper. If in the proper stochastic matrix P unity is a simple root of the characteristic polynomial, then the matrix P and the corresponding Markov chain are termed regular.

The following proposition is valid [14].

Theorem 2. The Markov chain C with a finite number of states has a limit distribution if and only if it is proper. In order that the chain C satisfy the property of ergodicity it is necessary and sufficient that it be a regular chain.

For the case of the regular finite Markov chain the limit transition probabilities are determined by the equations

$$p_{ij}^{\infty} = \frac{P_{ji}(\lambda)}{\psi_1(\lambda)} \quad (i, j = 1, 2, \ldots, n). \tag{74}$$

These equations are obtained from the special Perron equation (72) as the result of the limit transition as $k \to \infty$.

The results described above permit constructing the theory of the behavior of automata (random and deterministic) in random media. We shall limit ourselves to the consideration of only the Moore automata, since in the case of the Mealy automata there arises the necessity for certain complications of the theory which make it less easily visualized. We also agree to consider the deterministic automata as a particular case of the random automata, which, as mentioned above, is always possible.

With these assumptions every automaton A can be specified by the matrix $L = \|\lambda_{ij}\|$ of the output probabilities and by the family of matrices $D^{(m)} = \|\delta_{ik}^{(m)}\|$ of the transition probabilities. Any element λ_{ij} of the first matrix is equal to the probability of the appearance of the jth output signal in the case when the automaton A is in the ith state. The quantity $\delta_{ik}^{(m)}$ is the probability of the transition of the automaton from the ith state into the kth under the influence of the mth input signal.

The medium is specified for some class of automata having identical sets of input signals $(x_1, x_2, \ldots, x_n)$ and identical sets of output signals $(v_1, v_2, \ldots, v_s)$. Specification of the medium for the considered class K means the specification of the dependence of the input signal $x(t)$ of any automaton A from the class K at the arbitrary moment of discrete automaton time _t_ on the value of $v(t-1)$ of its output signal at the moment of time directly preceding the considered moment of time _t_. It is assumed that this dependence is the same for all the automata from the given K. In other words, the behavior of the medium is determined only by the operations (output signals) of the automata and does not depend directly on the internal arrangement of the automata.

Let us consider the _random media_ in which there are defined the so-called _reaction probabilities_ r_{jm} which are combined into the (rectangular) _reaction probability matrix_ $R = \|r_{jm}\|$. The value of r_{jm} is taken to be equal to the probability of the appearance of the m_th_ input signal at the input of the automaton A (from the class K) operating in the considered medium if in the directly preceding moment of time there was delivered by the automaton A the j_th_ output signal.

If the medium reaction probabilities are constant, the corresponding medium is termed a _stationary random medium_. In the _nonstationary_ random media the reaction probabilities can change with time. Just as in the case of the automata, the deterministic media (with a rigorously defined functional relationship $x(t) = f(v(t-1))$ can be considered as a particular case of the random media.

It is easy to see that the study of the behavior of the Moore automata (both determinate and random) in stationary random media reduces to the study of uniform Markov chains whose states can be identified with the states of the considered automata. Actually, the state $a(t)$ of the automaton A at any moment of time uniquely defines the probabilities of the output signals $v(t)$ and, consequently, in view of the definition of the stationary random medium, also the probabilities of the input signals of the automaton in the directly following moment of time $t + 1$. The latter probabilities uniquely determine the probabilities of the transitions of the automaton A from the state a(t) into any of the following states.

With the notations assumed above, the transition probabilities p_{ik} of the corresponding Markov chain are determined by the equations

$$p_{ik} = \sum_{l} \sum_{m} \lambda_{il} r_{lm} \delta_{ik}^{(m)}. \qquad (75)$$

We shall describe, following Tsetlin [82], several very simple problems on the behavior of automata in random media. To do this we consider the class K of determinate Moore automata having the two input signals $x_0 = 0$ and $x_1 = 1$ and the two output signals $v_0 = 0$ and $v_1 = 1$. We specify the stationary random medium C by the matrix R of the reaction probabilities

$$R = \left\| \begin{matrix} 1-p_0 & p_0 \\ 1-p_1 & p_1 \end{matrix} \right\|.$$

Let us term the input signal x_1 penalty and the input signal x_0 – no-penalty. Then we can say that with the output of the signal v_0 the medium penalizes the automaton with the probability p_0, and with output of the signal v_1 – with the probability p_1.

Let us consider first the Moore automaton A with the two states $a_1 = 1$ and $a_2 = 2$, given by the matrix of the output probabilities $L = \left\| \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \right\|$ and the matrices of the transition probabilities $D^{(0)} = \left\| \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} \right\|$, $D^{(1)} = \left\| \begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} \right\|$. In other words, A is a determinate automaton which delivers in the first state the output signal $v_0 = 0$, in the second state – the output signal $v_1 = 1$, retaining its state under the influence of the input signal $x_0 = 0$, and changing to the opposite state under the influence of the input signal $x_1 = 1$.

In accordance with what we have said above, the functioning of the automaton A in the medium C is described by the uniform Markov chain M with the two states $a_1 = 1$ and $a_2 = 2$. From equation (75) we easily find the matrix P of the transition probabilities of this chain

$$P = \begin{Vmatrix} 1-p_0 & -p_0 \\ -p_1 & 1-p_1 \end{Vmatrix}.$$

The characteristic polynomial $P(\lambda) = \begin{vmatrix} \lambda-1+p_0 & -p_0 \\ -p_1 & \lambda-1+p_1 \end{vmatrix}$ of the matrix is equal to $\lambda^2 - \lambda\,(2 - p_0 - p_1) + 1 - p_0 - p_1$ and its eigenvalues are respectively $\lambda_1 = 1$ and $\lambda_2 = 1 - p_0 - p_1$. If both probabilities the modulus of the second eigen value λ_2 is less than unity and, by theorem 2, the chain M will be ergodic in this case.

The polynomial $\psi_1(\lambda)$ will obviously be equal to $\lambda - 1 + p_0 + p_1$, and after application of equations (74) we easily find the limit transition probabilities of the considered chain

$$p_{11}^{\infty} = p_{21}^{\infty} = \frac{P_{11}^{(1)}}{\psi_1(1)} = \frac{p_1}{p_0+p_1} = \frac{p_1}{p_0+p_1}$$

and

$$p_{12}^{\infty} = p_{22}^{\infty} = \frac{1-1+p_0}{p_0+p_1} = \frac{p_0}{p_0+p_1}.$$

Thus, with sufficiently long functioning in the medium C the automaton A, regardless of the choice of the initial state will with the probability $p_1/p_0 + p_1$ be in the first state, and with the probability $p_0/p_0 + p_1$ – in the second state. Since the penalty probability in the first state of the automaton is equal to p_0, and in the second state is equal to p_1, then the mathematical expectation S of penalty of the automaton A at each step (after sufficiently long preliminary functioning) is expressed by the equation

$$S = p_0\frac{p_1}{p_0+p_1} + p_1\frac{p_0}{p_0+p_1} = \frac{2p_0p_1}{p_0+p_1}.$$

With $p_0 \neq p_1$ the quantity S is strictly less than the mean penalty probability $S_0 = 1/2\ (p_0 + p_1)$. Actually,

$$S_0 - S = \frac{(p_0+p_1)^2 - 4p_0p_1}{2(p_0+p_1)} = \frac{(p_0-p_1)^2}{2(p_0+p_1)} > 0,$$

where equality is obviously achieved only when $p_0 = p_1$. Thus, the considered automaton A possesses purposeful behavior in the sense that when it is placed in any stationary random medium which differentiates its two possible reactions it tends to that behavior for which the penalty value is on the average less than for an automaton delivering with equal probabilities both of the output signals (reactions) which are possible for the automaton A.

Let us now select in place of the considered automaton A the automaton A_n with 2n states 1, 2, ..., n, n + 1, ..., 2n − 1, 2n. We assume that in the states 1, 2, ..., n it delivers the output signal $v_0 = 0$, and in all the remaining states it delivers the output signal $v_1 = 1$. Assume, further, that the transition table of the automaton A_n is written as

$x \backslash a$	1	2	3	...	$n-1$	n	$n+1$	$n+2$	$n+3$	...	$2n-1$	$2n$
0	1	1	2	...	$n-2$	$n-1$	$n+1$	$n+1$	$n+2$	...	$2n-2$	$2n-1$
1	2	3	4	...	n	$2n$	$n+2$	$n+3$	$n+4$	...	$2n$	n

To this table there corresponds the transition graph shown in Fig. 12.

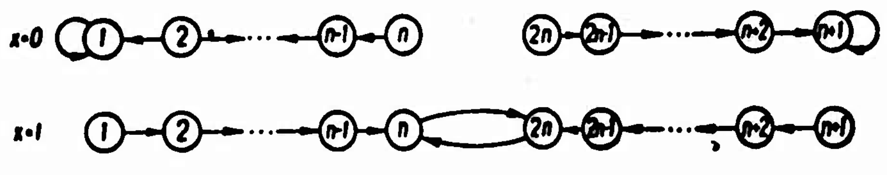

Fig. 12

From the form of its graph it is natural to term it an automaton with linear tactic. The automaton A analyzed above is obviously a particular case of the automaton A_n with linear tactic, for which the value of n is equal to unity. The behavior of the automaton A_n with linear tactic in the general case is studied exactly as in the considered particular case, although, of course, the corresponding operations are considerably more complex. These analyses lead to the conclusion that the value, calculated by analogy with the preceding case of the mathematical expectation S_n of penalty of the automaton A_n after one step of its operation (after a sufficiently long period of adaptation) with unlimited increase of the number n tends toward a natural minimal value S_{min}, equal to the smaller of the two numbers p_0, p_1. It is easy to see that the quantity S_{min} is the absolute minimum of the mathematical expectation of penalty for all automata operating in the considered random medium.

It is found that in many cases the apparatus of the uniform Markov chains can be used with success for the study of the behavior of automata not only in stationary, but also in certain nonstationary random media. Let us assume, for example, that there are several stationary ransom media C_1, C_2, ..., C_k similar to the medium C described above, but having different probability pairs p_0, p_1. From these media we can construct the nonstationary random medium N by introducing the matrix $B = \|b_{ij}\|$ of the transition probabilities of some Markov chain with k states. At any given moment t of discrete time the medium N acts like one of the media C_1, C_2, ..., C_k. If the model for its actions is the medium C_i, then we say that the medium N is in the ith state. The quantity b_{ij} is the probability of the transition of the medium N from the ith state into the jth $(i, j = 1, 2, \ldots, k)$. The probability b_{ij} is assumed to be constant and unchanging in the course of time.

If now some automaton A functions in the medium N, then the pairs (C_i, a_j), consisting of the state C_i of the medium N and the state a_j of the automaton A can be selected as the states of some uniform Markov chain. The matrix of the transition probabilities of this chain can be easily constructed from the matrices of the reaction probabilities of the media C_1, C_2, ..., C_k, the matrix B of the switching function and output function of the automaton A.

It can be shown [82] that in the class K of automata A_n with linear tactic operating in the described nonstationary medium N there is (depending on the choice of the medium N) an optimal automaton A_{n_0}, having a minimal (in the class K) mathematical expectation of the limiting value of the penalty at each step of its operation. Thus, in contrast with the stationary meida, for automata with linear tactic operating in nonstationary media, it is advisable to increase the volume of the automaton memory (number of states) only to a certain limit, after which further increase of the memory leads to deterioration rather than improvement of the quality of the operation of the automaton.

§5. THE PROBLEM OF PATTERN RECOGNITION TRAINING

One of the most significant fields of application of the theory of the self-organizing systems is that of the problem of the recognition of visual patterns. The recognition of visual patterns and the training for such recognition is a brilliant example of the adaptive properties of the human brain. The meaning of pattern recognition is that the human observer combines certain sets of objects or phenomena which he observes into a single class, termed the pattern. The patterns with which the human being operates are not random combinations of objects, but rather those combinations which are related by some common properties. Considering basically the visual patterns, we shall in the future term the individual objects which compose this pattern <u>images</u>.

Examples of visual patterns might be the set of all the images of a particular letter or digit, the set of the images of all possible buildings, etc. By analogy with the visual patterns we can consider also the sound patterns (for example, the set of all the pronounciations of a particular phonem, the set of all waltzes, etc.) and patterns of any other nature. In the future we shall limit ovrselves to the consideration of only the visual patterns, as examples however, all our theoretical constructions will be applicable not only for the visual, but also for any other patterns.

For the following constructions we need first of all the definitions of the abstract images and patterns. We will assume that the images are perceived by some set of sensitive elements – receptors. By analogy with the case of the visual patterns perceived by the human eye, we term this set the retina. In the case of abstract images and patterns we do not need to be more specific on the question of the spatial arrangement of the receptors. However, in the case of specific visual patterns this refinement is useful. In this case we will usually assume that the receptors constituting the retina are arranged on a plane at points forming a regular grid, i.e., in other words, at the points with the coordinates $(a + ic, b + jc)$, where $c \neq 0$, i runs through the set of values $0, 1, \ldots, m - 1$ and j runs through the set of values $0, 1, \ldots, n - 1$. In the future we shall such a retina a regular rectangular $(n \times m)$-retina.

The task of the retina is to convert the image projected onto it into some ensemble of signals of a standard form which are put out by the receptors composing retina. In the future we shall differentiate two forms of receptors: the so-called continuous receptors whose output signals can be any real numbers on some fixed segment of the number line, and the so-called discrete receptors which can deliver only two different output signals. Without losing generality we can fix as the domain of the values of the output signals of the continuous receptors the number segment $[0, 1]$, and as the possible values of the output signals of the discrete receptors we can fix the ends of this segment, i.e., the numbers 0 and 1.

In the case of the visual patterns we will always assume that the output signal of the continuous receptor is equal to the brighteness of the image point projected on the given receptor, expressed in relative units: zero corresponds to absolutely black points, and unity to absolutely white (reflecting 100% of the light incident on them) points of the image. For the discrete receptors we establish some brightness threshold. The points whose brightness does not exceed the value of this threshold will correspond to a zero output signal. More specifically, however, in the case of the discrete receptors we shall consider only two-tone images consisting either of points of zero brightness (background) or of points of unit brightness (the image itself). In the future we shall use precisely this latter point of view.

Absrtact image is the term we shall give to any fixed ensemble of output signals of the receptors constituting the retina. If the total number of receptors in the retina is N, then, in view of the assumptions made above, the abstract image can be naturally identified with some point of an N-dimensional unit cube. In the case of continuous receptors all the points of this cube correspond to the images, while in the case of the discrete images only the cube vertices correspond to the images. In connection with this, we shall call the N-dimensional unit cube (in the case of the continuous receptors) or the set of its vertices (in the case of the discrete receptors) the image space. In the first case this space is continuous, in the second case it is discrete (consisting of 2^N different points).

It is natural to introduce the following "metric" in the image space: the distance between two points of this space (i.e., between two images) is the square root of the sum of the squares of the differences of the corresponding coordinates of these points

$$d = \sqrt{(x_1' - x_1'')^2 + (x_2' - x_2'')^2 + \ldots + (x_N' - x_N'')^2}.$$

The R-neighborhood of any point M of the image space is the ensemble of all points of this space removed from the point M by a distance less than or equal to R.

We note that the definitions introduced are applicable for both the continuous and the discrete image spaces. In the second case the distance between any two points is the square root of the total number $\underline{n}$ of the noncoincidences of the coordinates of these points. However, it is more convenient for us to consider that in the case of the discrete spaces the distance is this number $\underline{n}$ itself. Then all the distances will be expressed by whole numbers. After the introduction of the distance in the image space, we can talk of the closeness of particular points to one another. From the intuitive considerations associated with the concept of the pattern, it follows that the images lying sufficiently close to a particular image from some pattern must belong to this pattern itself. This circumstance must be somehow taken into account in the definition of the concept of the abstract pattern.

In the case of the continuous image space, the pattern can be only that set of points of this space which together with any point M also wholly contains some ε-neighborhood of the point M (the magnitude of ε depends on the choice of the point M). Sets having this property are termed open sets. Thus, in the case of the continuous receptors we shall term any open set of the image space an abstract pattern.

An example of an open set might be the internal portion of a sphere having the same dimension as the considered (continuous) image space. It is important to once again emphasize that the degree of smallness of the changes which can be introduced in an image without changing its membership in a given pattern depends on the choice of the image itself. The permissible variations for the images located closer to the boundary of the pattern (in the example in question the surface of the sphere serves as the boundary) are reduced, while they are increased for the images sufficiently removed from the boundary.

In the case of the discrete image space all its subsets will obviously be open sets and can consequently be considered as abstract patterns. For reducing the classes of patterns in the discrete space we can make use of the concept of the boundary index of the set. Let M be any subset of the discrete image space R which we introduced, and m the number of elements of this set. If the set M does not coincide with the entire space R, then among its elements there will be those for which at a distance from them equal to unity there lie elements not belonging to the set M. Let us term such elements boundary elements and denote by m_1 the number of all the boundary elements of the set M. We call the ratio m_1/m the boundary index of the considered set M.

It is easy to see that the smaller the boundary index of a set the greater the degree to which it resembles in its properties the open sets of the continuous spaces: an ever larger portion of the points of the set with their 1-neighborhoods are contained in it. Therefore it is natural to state the proposition which Braverman [11] has termed the compactness hypothesis: only those sets whose boundary indices are sufficiently small can serve as patterns in the discrete image space.

With a more detailed study it is found that this proposition must be refined by means of certain additional probability-theoretic constructions. Let R be a discrete image space in which some subset has been fixed. Let, further, for each element x_1 of the space R there be given the probability $f(x_1)$ of the appearance of this element (of the image) in some series of experiments of the type of independent trials; let $f_{S}(x_1)$ be the conditional probability of the appearance of the element x_1 with the condition that it belongs to the pattern S. If the element x_1 belongs to the pattern S, then for any natural number k $Q_k(x_1)$ is the set of all points not contained in the pattern S and removed from the element [illegible] a distance less than or equal to k. The quantity $g_S(k(= \sum_{x_i \in S} f_S(x_i, \ldots x_j \to [illegible] (x_i))$ is the probability or wrong assignment in a following trial of an element of the pattern S which does not belong to it as a result of the inclusion in S of all elements lying in the k-neighborhood of the element x_1 in the preceding trial in which some element x_1 from the pattern S was randomly selected.

We term the operation of the inclusion in a particular pattern S of all elements of the k-neighborhood of some element x from this pattern the operation of k-extrapolation with respect to the element x. The quantity found above $g_S(k)$ is the probability of the occurrence of an error as the result of the operation of k-extrapolation with respect to the randomly selected element of the pattern S. A refinement of the compactness hypothesis, mentioned above, consists in the assumption that for every discrete image space R there exists such a number N = = N(R) that $N \geq 1$, and for all values of $k \leq N$ the probability $g_S(k)$ of the occurrence of an error in the result of the k-extrapolation does not exceed the negligibly small constant nonnegative quantity ε for any patterns S. We term this the hypothesis of the N-extrapolatability of the patterns with accuracy to ε.

The operation of k-extrapolation can obviously also be defined for patterns in continuous image spaces. Replacing the summing by integration, we can obtain by analogy with the expression for $g_S(k)$ an expression for the probability of the appearance of an error as the result of extrapolation in the continuous case. In exactly the same way as for the discrete spaces, we can formulate the hypothesis of the N-extrapolatability of the patterns in the continuous image spaces.

The pattern recognition problem leads to the need for a precise description of the features characterizing this pattern. However, this sort of description cannot be given in all cases by any means without overcoming very serious difficulties. Therefore, in practice we usually follow the path of constructing algorithms which make it possible to accomplish the so-called <u>training</u> for pattern recognition. The essence of this training is obtaining the approximate descripton (or, as customarily phrased, the approximation) of the pattern as the result of the showing of some set (generally speaking, not all!) of images composing this pattern.

Based on the hypothesis on the N-extrapolatability of the patterns, we can construct the so-called <u>general approximation algorithm</u> which makes it possible to accomplish pattern recognition training. Just as every algorithm of the self-improving type, the general approximation algorithm A has two operation periods – the learning period and the examination period.

In the learning period various representations of the patterns R_1, R_2, ..., R_n, which are to be recognized are applied to the input of the algorithm A. In this case the corresponding representations (images) are chosen at random (most frequently by the method of independent trials) and are accompanied by the indication: to which of the patterns R_1, R_2, ..., R_n each of the selected images belongs. All the images shown in the learning period are stored and are used in the examination regime for the determination of whether the next (also selected randomly) image $\underline{r}$ belongs to a particular one of the pattern R_1, R_2, ..., R_n.

To do this determinations are made of the distances in the image space from the image $\underline{r}$, first to the representations of the pattern R_1 selected in the learning period, and then to the representations of the pattern R_2, etc., until the succeeding distance determined to some representative of some pattern $R_i (i = 1, 2, ..., n)$ is found to be less than or equal to the extrapolatability coefficient N. In this case the image $\underline{r}$ is associated with the pattern R_1. If, however, all the distances are larger than N, the image $\underline{r}$ remains unrecognized, i.e., it will not be associated with any of the patterns R_1, R_2, ..., R_n.

It is easy to see that if all the patterns R_1, R_2, ..., R_n are extrapolatable with an accuracy to ε and can be covered with the aid of a number of N-neighborhoods (spheres of radius N) which is significantly less than $1/\varepsilon$, then the described algorithm gives a good approximation of the chosen patterns (with small probability of error on examination).

In practice the different patterns in the image space are not usually in direct contact with one another. If we take as the coefficient N the minimal distance between patterns, then in the discrete space the absence of contact of the patterns means that $N \geq 2$. If we also assume that the probability of the appearance of images not belonging to any one of tne selected patterns R_1, R_2. ..., R_n, is equal to zero, then all these patterns are obviously $(N - 1)$-extrapolatable with an accuracy to $\varepsilon = 0$. With these assumptions the general algorithm for approximation with the aid of (N - 1-neighborhoods obviously leads, as a result of a sufficiently long duration of the learning period, to an arbitrarily good approximation (with an arbitrarily small error probability).

This general approximation algorithm admits several further improvements in several different directions. First, in addition to the examination regime described above we can introduce a second type of examination regime. In this case the image appearing in the examination regime relates to that one of the patterns R_1, R_2, ..., R_n, which contains the representative (memorized in the learning period) located closest of all to the image $\underline{r}$. For definiteness we assume that if there are several such patterns preference is given to that one of them which has the smallest number.

The second improvement consists in economy of the memory: if the N-neighborhood of any image S, appearing in the learning period, is completely covered by the N-neighborhoods of the other images, also shown in the training period, then the image S can immediately be eliminated from the memory and not used during the examination. In practice it is advisable to make use of this improvement in a somewhat different modification in which the maximal number of represen'ations of each pattern which can be remembered in the learning period is limited ahead of time. For each remembered representation account is taken of its relative usefulness. As the criterion of the relative usefulness of any image r we can use, for example, the ration of the number of cases when this image was used for the correct recognition of the following images to the total number of images which appeared after memorization of the image r. Only those images are subject to memorization which cannot be correctly recognized with the aid of the images already available in the memory. If, in addition, the memory set aside for the storage of the representations of a particular pattern is found to be completely full, then the representation being memorized forces out of the memory the representation of the given pattern which has the lowest relative usefulness.

The third improvement involves, along with the "natural" retina (onto which the images being recognized are projected directly), the use of a new retina whose output signals are suitably selected functions of the output signals of the first retina. The image space in which the patterns are defined is constructed from the output signals of the second retina (incidentally, it is more natural to call this the feature space rather than the image space). With the aid of a judiciously chosen transformation of the original space we can considerably simplify the problem of pattern recognition training. Commonly used, in particular, are those transformations which generate features which are not dependent on parallel shift or variation of the size of the images.

Finally, we will indicate still another modification of the general approximation algorithm A. The regime in which this algorithm receives during the period of its self-improvement certain images with an indication of the pattern to which they belong is termed the training regime. In many cases, in addition to this regime it is advisable to consider the so-called self-training regime.

In the self-training regime planned for the formation of n different patterns, there are first given the n randomly selected images r_1, r_2, ..., r_n, each of which is taken to be the representation of some pattern. The image s_1 being reshown is associated to that pattern whose representation is located closest to the image s_1 and the number of representatives of this pattern is increased. In the following step the reappearing image s_2 is compared with the augmented number of representations and is again associated with that pattern whose presentation (any) is located closer to s_2 than the representatives of all the other patterns. Thereafter the storage proceeds either according to the usual scheme or by the scheme described above with replacement (with limited memory). The recognition of the images in the examination regime can be performed by the two methods described above: either by N-extrapolation of the patterns, or by the method based on the determination of the shortest distance.

This algorithm can be used in the case of both the discrete and continuous image spaces. The approximation method used in it is unique, of course. Thus, rather than approximation by the spherical neighborhoods we could use neighborhoods of any other shape for the approximation. Successful results have been obtained with approximation of the patterns by regions bounded by hyperplanes (see, for example, Braveman [11]). Various modifications of the metric described above are also possible in the image space, which obviously leads to alteration of the concept of the spherical neighborhoods: neighborhoods which are spherical in one metric may not be so in another metric, and vice versa.

We shall describe several other, more specialized algorithms for pattern recognition training which have been used successfully by various authors. One of the simplest, although not extremely effective, algorithms of this sort is the so-called perceptron of Rosenblatt [69]. Just as every device for the recognition of patterns, the perceptron contains a set of receptors – the retina. In the future, without specially stipulating this each time, we shall consider only regular rectangular retinas. Depending on the nature of the receptors, the perceptrons are divided into the continuous perceptrons (with continuous receptors) and the discrete perceptrons (with discrete binary receptors).

In addition to the receptors, each perceptron contains two other forms of element, termed A-elements and R-elements.

The A-elements are simplified models of the neurons. In this connection we shall hereafter term them simply neurons. In accordance with the nature of the receptors used in the perceptron we differentiate the continuous and discrete neurons. Both types of neurons have two forms of inputs, termed stimulating and inhibiting. Each neuron has a finite number of inputs and a single output; in addition, with it there is associated some real number, termed the weight of the given neuron. As the domain of the values of the neuron weights we take the set of all real numbers, regardless of which neurons we are considering – continuous of discrete.

In addition to the weight, the number of stimulating and the number of inhibiting inputs, the neuron is also characterized by its functioning law, which determines the output signal of the neuron as a function of its input signals and weight. We must keep in mind that the inputs of all the neurons in the perceptron are connected to the receptors, so that the signals generated by the receptors serve as the input signals for the neurons.

The continuous neurons, first considered by Rosenblatt [69], had a functioning law of the form $z = v(\Sigma x - \Sigma y)$, where z is the output signal, v is the neuron weight, Σx is the sum of the signals applied to the neuron through the stimulating inputs, and Σy is the sum of the signals applied to the neuron through the inhibiting inputs (x, y, v and z are arbitrary numbers).

The functioning law of the discrete neurons is normally specified by the indication of some whole rational number p termed the neuron triggering threshold, or simply threshold. If the algebraic sum $\Sigma x - \Sigma y$ of the stimulating and inhibiting input signals is less than the threshold, then the neuron is considered unstimulated and delivers an output signal equal to zero. When the sum $\Sigma x - \Sigma y$ reaches and exceeds the threshold, the neuron is stimulated and delivers an output signal equal to its weight v (regardless of the magnitude of the amount by which the sum of the input signals exceeds the threshold).

It is convenient to characterize the discrete neurons with the described functioning law by means of three whole numbers (k, ℓ, p), the first being equal to the number of stimulating inputs, the second to the number of inhibiting inputs, and the third to the threshold level. In the following considerations the neuron weight will always be a variable quantity and therefore we shall not introduce it into the neuron characteristic. Discrete neurons of the indicated type, having the same characteristic three numbers (k, ℓ, p) will be associated with the same type, regardless of possible differences of their weights.

Hereafter it is assumed that all the neurons of any given perceptron designed for the differentiation of k different patterns, the set of all the neurons is partitioned into k disjoint groups (subsets) located in one-to-one correspondence to the pattern being distinguished. For brevity we shall term the neurons belonging to the group corresponding to the ith pattern the neurons of the ith pattern (i = 1, 2, ..., ..., k).

The inputs of each neuron in the perceptron are connected to the receptors of the retina. Here it is assumed that the different inputs of the same neuron are connected to different receptors. The outputs of the neurons are connected to special summators termed R-elements, with the outputs of the neurons of the same pattern connected to the same summator, termed the summator of this pattern.

The output signal of the summator of any given pattern is equal to the sum of the weights of all the stimulated neurons of this pattern. If none of the neurons of the pattern being considered is stimulated, then the output signal of the corresponding summator is taken equal to zero. The final output signal of the entire perceptron is considered to be that pattern whose summator has the highest output signal. In the case when the maximal value of the output signal is attained simultaneously by the summators of several patterns, the output signal of the perceptron is considered to be undefined.

Taking as the input signal of the entire perceptron the image being projected on its retina, we obtain as the reaction of the perceptron to this signal that pattern to which the perceptron relates the given image. It does not follow at all, of course, that the considered perceptron accomplishes the proper classification of the images in accordance with an a priori specified division of the set of images into different patterns. This initial division is specified by the operator. We shall term it the original (or a priori) classification of the images in contrast with the actual classification accomplished by the chosen perceptron.

Therefore it is necessary also to specify some process of variation of the perceptron characteristics which permits approach of the actual classification performed by the perceptron to the original classification as we show the perceptron various images. This process is specified with the aid of the indication of the so-called encouragement law.

As the basic encouragement law for the discrete perceptrons we shall choose the somewhat generalized encouragement law in the so-called α-systems which were considered by Joseph [34]. This law, which we shall term the (generalized) α-law, is completely characterized by the specification of two nonnegative constants a and b, not simultaneously equal to zero. The meaning of this encouragement law consists in the weights of some neurons being increased by an amount equal to a and the weights of the others being decreased by an amount equal to b after each showing of a succeeding image to the perceptron (the encouragement law in the Joseph α-systems is obtained from the generalized α-law in the case when $a = 1$, $b = 0$).

We differentiate two regimes of functioning of the perceptron with generalized α-law encouragement. The first regime, termed the training regime, consists in the encouragement (increase of weight by the amount a) of all the stimulated neurons of that pattern to which the image being considered in the given step belongs, and in the penalizing (reduction of the weight by the amount b) of all the stimulated neurons of the remaining patterns. It is clear that the correct pattern to which the given image belongs must be indicated by the human teacher, since only he knows the original a priori classification of the images.

The second regime, termed the self-training regime, differs from the training regime only in that the determination of the patteren to which the image being considered belongs is accomplished by the perceptron itself – this pattern is taken to be that pattern which actually was delivered by the perceptron in response to the showing of the given image. Of course, here there is no guarantee that the response delivered by the perceptron will be correct (in the sense of the original classification of the images). However, with observation of certain conditions, in the case of unlimited increase of the number of steps in the self-training process the process can sometimes reproduce the original classification of the images.

In addition to the (generalized) α-law encouragement in several cases it is advisable to consider two other laws, which we shall term respectively the (generalized) β-law and the (generalized) γ-law. Both of these laws retain the priciple of encouragement and penalizing which is used in the (generalized) α-law. In addition to this, in the β-law at each step (in both the training and self-training regimes) there is a reduction of the weight of all the neurons (both stimulated) by an amount which is directly proportional to their weights, with a proportionality coefficient β which is the same for all the neurons. In the γ-law there is performed an additional (to the operations of the α-law) variation of the weights of all the neurons (both stimulated and unstimulated) by the same amount, selected at each step so that the sum of the weights of all the neurons is always equal to zero.

In the case of the continuous neurons the generalized α-law of encouragement consists in that any neuron of the correct (a priori or from the point of view of the perceptron) pattern increases its weight by the value of the product q of the constant a by the combined input signal of the neuron: $q = a(\Sigma x - \Sigma y)$. Similarly, the weights of the neurons of all the remaining patterns are reduced by the amount $b(\Sigma x - \Sigma y)$ (individually for each individual neuron). The additions which differentiate the β- and γ-laws remain the same as in the discrete case.

In the construction of the theory of perceptron training and self-training it is frequently advisable to consider not the individual perceptrons, but certain classes of perceptrons. A perceptron class is a set of perceptrons which can differ from one another only in the method of connection of the neurons with the receptors and the initial weights of the neurons. All the remaining characteristics of the perceptrons belonging to a particular class are assumed to be the same. These characteristics include the form of the receptors and neurons, the total number of receptors and the structure of the retina, the set of images and the set of patterns, the original classification of the images (their distribution over the patterns), the number of neurons of each pattern, and, finally, the encouragement law.

The method of connecting the neurons with the receptors and the initial weights of the neurons are considered random and are characterized (within the limits of the selected class) by certain distribution laws. In other words, the class of perceptrons is considered not as an abstract set of perceptrons, but as a set with specified probability field which determines the probability of the selection of a particular concrete representation of the class being considered. We can thus consider that the specification of the class defines some random perceptron.

The initial weights of the neurons are usually considered to be independent random quantities having the same distribution law. In the same way the method of connection of each neuron with the retina is assumed to be independent of the connections of the remaining neurons. To each possible method of connection of an individual neuron with the retina there is associated the probability of this method, common for all the neurons. Here the connection of all the neurons of the perceptron (from the perceptron class being considered) to the retina is treated as a series of independent trials, characterized by the indicated probabilities.

Combining the probability characteristic for the method of connecting the neurons with the retina with the distribution law for the initial weights of the neurons, we arrive at the desired distribution law in the class of perceptrons. One of the most frequently encountered distribution laws is obtained when all the initial weights are determined and are equal to the same number (most frequently zero), and the connection of all the inputs of any given neuron is accomplished independently of one another on the basis of a particular distribution law (most frequently uniform) specified directly on the retina.

In the construction of the perceptron training theory we must consider the so-called training sequences and the classes of training sequences. The training sequence is simply a finite sequence of images, shown to the perceptron one after another in the process of its training or self-training. The total number of images shown (including repetitions) is termed the length of the training sequence. A class of traning sequences is the set of all sequences of the same length in which there is given the distribution law which defines the probability of the selection of any given sequence of the considered class.

Most frequently this distribution law is obtained with the assignment of a definite value of the probability of the appearance of any image from the considered set of images at each step of the training, where we usually consider the case when these probabilities are identical in all the steps, i.e., when the training sequence is a series of independent experiments on the selection of the images with constant probabilities assigned to each image. Hereafter we shall limit ourselves to this case.

The effectiveness of the training in a given class A of perceptrons with the aid of the given class B of training sequences is defined as the probability of correct recognition of the next image p applied to a perceptron randomly selected from the class A after the preliminary application to it of a training sequence randomly selected from the class B. We differentiate two forms of effectiveness. The so-called total effectiveness of training is obtained when the image p is selected at random (with the a priori fixed probabilities of the appearance of the various images used in the establishment of the distribution law in the class of training sequences). The training effectiveness with respect to the single image q is obtained when the next image presented to the perceptron to recognize is precisely the image q . If the probabilities of recognition errors are the same for all the images then the total training effectiveness will obviously coincide with the inidvidual effectiveness of training with respect to any image.

In the following section we shall undertake the theoretical study of the training effectiveness for discrete perceptrons with α-law encouragement. For the moment, we note that experiments have shown that the training effectiveness in all the types of perceptrons described above is relatively low. Therefore in the algorithms for pattern recognition training which are used in practice there are normally introduced several additional improvements in comparison with the perceptron scheme. For example, in the scheme of the Roberts' adapt [67], on the whole quite similar to the scheme of the perceptron with α-law encouragement, a considerable improvement of training effectiveness is acheived by preliminary normalization of the image (i.e., in other words, by automatic shift of the image to the center of the retina and its reduction to a standard size). Methods are also used for the preliminary processing of features, schemes of multi-stage perceptrons, etc.

The deficiencies of the perceptron and the ways of removing them will be clearer after acquaintance with the following two sections in which we consider some questions associated with its behavior in the training and self-training regimes.

§6. THEORY OF TRAINING OF DISCRETE α-PERECPTRONS

In the present section we shall note the basic outlines of the theory of perceptron training. Here we shall limit ourselves to the consideration of only the discrete perceptrons with generalized α-law encouragement operating in the training regime (and not self-training!), without special stipulation of this circumstance in each case. In this case the training theory is more simple and transparent, since it is possible to follow not the functioning of each individual neuron, but to limit ourselves to the consideration of only certain integral characteristics.

In the variant of the theory which we assume, this integral characteristic is the so-called characteristic tensor of the perceptron. We immediately emphasize that the use of the term "tensor" in this case is not related with any patterns of transformation of its component with variation of the coordinate system, but serves only as the name for a certain integral table with three inputs. For the description of this table we introduce a definite numeration of all the images which are being presented to the considered perceptron by the numbers from 1 to $\underline{m}$ and the numeration of all the patterns into which these images are subdivided by the numbers from 1 to $\underline{q}$. Then the characteristic tensor of the perceptron will be the ensemble of components T^k_{ij}, where the indices i and j run through the values from 1 to m and the index k runs through the values from 1 to q. T^k_{ij} denotes the number of neurons of the kth pattern which are stimulated by both the ith and the jth images.

The characteristic tensor of a class of perceptrons is defined essentially the same way. The only difference is that its components T^k_{ij} will in this case be not determinate, but random quantities whose distribution laws are determined in an obvious fashion by the distribution law which characterizes the method of connection of the neurons to the retina.

These definitions imply the validity of the relation

$$T^k_{ij} = T^k_{ji} \quad (i, j = 1, 2, \ldots, m;\ k = 1, 2, \ldots, q). \tag{76}$$

It is also clear that any "diagonal" element of the tensor, T^k_{ii} for example, is the number of neurons of a particular (the kth in the present case) pattern which are stimulated under the action of the ith image. This implies the validity of the inequaltiy

$$T^k_{ii} > T^k_{ij} \quad (i, j = 1, 2, \ldots, m;\ k = 1, 2, \ldots\ q). \tag{77}$$

We shall term a perceptron or class of perceptrons symmetrical if the components of the characteristic tensor do not depend on the upper index, i.e., if the following relation is valid

$$T^{k_1}_{ij} = T^{k_2}_{ij}\ (k_1, k_2 = 1, 2, \ldots, q;\ i, j = 1, 2 \ldots m). \tag{78}$$

In this case the upper index is redundant so that it is natural to characterize the symmetrical perceptrons and the classes of perceptrons not by the three-input (T^k_{ij}) but by the two-input table (T_{ij}) where $T_{ij} = T^1_{ij} = T^2_{ij} = \ldots = T^q_{ij}$, which we shall term the characteristic matrix of the perceptron (or class of perceptrons).

Let us introduce still another notation. For any (finite) sequence of images ℓ we use $U_1^k (\ell)$ to denote the output signal of the summator of the kth pattern which is induced in the considered perceptron by the ith image shown after training of the perceptron with the training sequence ℓ. We use U_1^k to denote the corresponding signal prior to the beginning of the training process, i.e., in other words, the signal $U_1^k(\ell)$ for the case when the training sequence ℓ is empty (has a length equal to zero).

The quantities $U_1^k(\ell)$ will obviously be determinate in the case of the selection of a particular perceptron and random in the case of the consideration of a class of perceptrons. Sometimes it is advisable also to consider the sequence ℓ as a random sequence, running through the class of training sequences.

A distinctive feature of the α-law for perceptron training is the unique property of commutativity of the training process expressed by the following proposition.

Theorem 1. In the perceptron (or in a class of perceptrons) with α-law encouragement the output signal $U_1^k(\ell)$ of the summator of the kth pattern does not change under the action of the ith image after training with any sequence ℓ if in the sequence ℓ there is performed an arbitrary permutation of the images composing it. This is valid for any image i and any pattern k.

Actually, the input signals of the neurons of the kth pattern which are induced by the ith image will obviously not be altered in the training process, so that they remain the same after showing of the training sequence ℓ and any other sequence ℓ'. Thus, the variation of the output signal of the summator in the training process is due only to the change of the weights of the neurons. As a result of the definition of generalized α-law encouragement, the variations of the weights of the neurons with the showing of any image in the training process (but, generally speaking, not in the self-training process) do not depend on the place which the given image occupies in the training sequence. Since the overall increment of the weight of any neuron in the training process is equal simply to the sum of the increments at each step of the process, the validity of theorem 1 is thereby completely proved.

By the use of theorem 1 we can characterize any training sequence ℓ by the integral vector $\vec{v} = (v_1, v_2, \ldots, v_m)$ whose ith component (for any $i = 1, 2, \ldots, m$) is equal to the number of occurrences of the ith image in the sequence ℓ. Let us call this vector the characteristic vector of the sequence ℓ. The class of training sequences can also be specified with the aid of the characteristic vector. However, the components of the vector in this case will be, generally speaking, not determinate, but random quantities.

For the description of the perceptron training process with (generalized) α-law encouragement it is sufficient to specify the original perceptron (or class of perceptrons) by only its characteristic tensor (T^k_{ij}) and the matrix of the initial signals of the pattern summators (U^k_i) $(i, j = 1, 2, \ldots, m; k = 1, 2, \ldots, m)$. The training sequence ℓ (or class of training sequences) is specified by its characteristic vector $(v_1, v_2, \ldots, v_n)$. In the general case all the quantities T^k_{ij}, U^k_i, v_i will be random. However, most frequently we consider various particular cases when certain of the indicated quantities are determinate. We note that we will usually include an indication on the seleaction of the image set and the training sequences in the definition of the perceptron.

In order to avoid confusion of the images and the patterns, we shall designate the patterns by Latin letters and the images as before by their numbers. Let us consider the question of the determination of the output signals of the pattern summators $U_i^P(\ell)$. It is easy to see that with the use of (generalized) α-law encouragement the quantity $U_i^P(\ell)$ is represented in the form of the initial signal U_i^P and the increments of the weights of all the neurons of the Pth pattern which are stimulated by the ith image all the steps of the training process.

Characterizing the class of the training sequences by the characteristic vector $(v_1, v_2, \ldots, v_m)$, it is not difficult to find the expression for the overall increment of the quantity $U_i^P(\ell)$ obtained as the result of v_j showings of the jth image. It follows from the definition of (generalized) α-law encouragement with the constants a and b that with each showing of the jth image any neuron of the Pth pattern which stimulates this image will increase its weight by the amount a if $j \in P$, and will reduce its weight by the amount b if $j \bar{\in} P$. The total number of neurons of the Pth pattern which participate in the formation of the output signal $U_i^P(\ell)$ and which stimulate the jth image is clearly equal to T_{ij}^P. Thus, the total increment of the magnitude as the result of v_j showings of the jth image is expressed by the formula $aT_{ij}^P v_j$, if $j \in P$, and by the formula $bT_{ij}^P v_j$ if $j \bar{\in} P$. Therefore the following proposition is valid.

Theorem 2. Let there be given the discrete perceptron (or class of discrete perceptrons) with the characteristic tensor T_{ij}^P and the matrix of initial output signals of the pattern summators U_i^P $(i, j = 1, 2, \ldots, m;\ P \in R)$. If in the considered perceptron (class of perceptrons) there operates the (generalized) α-law encouragement with the constants a, b, then after training with the sequence (or class of sequences) ℓ with the characteristic vector $(v_1, v_2, \ldots, v_m)$ for any pattern P and any image i the output signal $U_i^P(\ell)$ of the summator of the Pth pattern under the action of the ith image is expressed by the equation

$$U_i^P(l) = U_i^P + a\sum_{j\in P} T_{ij}^P v_j - b\sum_{j\bar{\in} P} T_{ij}^P v_j. \qquad (79)$$

For any image $\underline{i}$ we shall use P_i to denote that pattern to which the image $\underline{i}$ belongs in the original classification of the images. Using this notation, it is not difficult to write out the necessary and sufficient condition for the perceptron to correctly classify the $\underline{ith}$ image after training. This condition will obviously be the satisfaction of the inequality

$$U_i^{P_i}(l) > U_i^P(l) \text{ for all } P \neq P_i. \qquad (80)$$

Using relations (79) and (80) it is not difficult to calculate the perceptron training effectiveness in any specific case. These relations take a particularly simple form in the case of the symmetrical perceptrons. Actually, since in this case $T_{ij}^{Pi} = T_{ij}^P = T_{ij}$, relations (79) and (80) can be written in the form of the system of inequalities

$$U_i^{P_i} + a\sum_{j\in P_i} T_{ij}v_j - b\sum_{j\bar{\in} P_i} T_{ij}v_j > U_i^P + a\sum_{j\in P} T_{ij}v_j - b\sum_{j\bar{\in} P} T_{ij}v_j. \qquad (81)$$

In inequality (81) terms of the form $b\Sigma T_{ij}\ v_j$, for which j is not contained in either P_i nor P, appear in both the left and right sides and therefore cancel one another. After their exclusion we obtain the simpler relations equivalent to relation (81):

$$U_i^{P_i} + (a+b)\sum_{j\in P_i} T_{ij}v_j > U_i^P + (a+b)\sum_{j\in P} T_{ij}v_j \text{ для всех } P \neq P_i. \qquad (82)$$

Inequalities (82) give the necessary and sufficient conditions for the correct classification of the $\underline{ith}$ image by a $\underline{symmetric}$ perceptron with the characteristic matrix $\|T_{ij}\|$ and the initial signals of the pattern summators U_i^P after training with the sequence having the characteristic vector $(v_1, v_2, \ldots, v_m)$. These inequalities can be simplified still more for the perceptrons with $\underline{\text{symmetrical initial conditions}}$, i.e., those percpetrons (or classes of perceptrons) for which the conditions

$$U_i^P = U_i^Q \quad \text{for all } i = 1, 2, \dots, m \tag{83}$$

are satisfied for all P and Q.

Using relations (83) and recalling that as the result of the definition of the generalized α-law $a + b \neq 0$, we come to the following result.

Theorem 3. Let there be given any discrete symmetric perceptrons) with the characteristic matrix $\|T_{ij}\|$ and with symmetric initial conditions in which there operates (generalized) α-law encouragement. Then the necessary and sufficient conditions for the correct recognition by the perception (class of perceptrons) of any ith image after training with the sequence (class of sequences) with the characteristic vector $v_1, v_2, \dots, v_m$) is expressed by the relations

$$\sum_{i \in P_i} T_{ij} v_i > \sum_{i \in P} T_{ij} v_i \quad \text{for all } P \neq P_1.$$

Corollary: training effectiveness in symmetrical discrete perceptrons with symmetric initial conditions with performance in them of (generalized) α-law encouragement does not depend on the selection of the (nonnegative) constants a and b which characterize the law.

Thus, in the study of the symmetric discrete perceptrons with symmetric initial conditions we can without losing generality use conventional α-law encouragement with the constants (1, 0) rather than the generalized α-law with the constants (a, b).

In specific calculations of training effectiveness in classes of perceptrons it is usually assumed that all the neurons are connected to the retina independently from one another, and the probability α_{ij} of such a connection of the neuron that it will be stimulated by both the ith and the jth image is the same for all the neurons with any fixed values of i and j.

If we use T^P to denote the total number of neurons of the Pth pattern, then the component T^P_{ij} of the characteristic tensor of the class of perceptrons being considered can be treated as the number of occurrences of some event having the probability α_{ij} with T^P independent trials. As a result of theorem 2 from §2 of the present chapter, the mathematical expectation $E(T^P_{ij})$ and the variance $D(T^P_{ij})$ of the random quantity T^P_{ij} are expressed by the equations

$$E(T^P_{ij}) = T^P\alpha_{ij};\ D(T^P_{ij}) = T^P\alpha_{ij}(1-\alpha_{ij}) \qquad (85)$$
$$(i, j = 1, 2, \dots, m;\ P \in R).$$

With sufficiently large values of T^P the quantity T^P_{ij} itself can be considered normally distributed. We note also that in the case of the symmetrical perceptrons the quantities T^P will be equal to one another for different patterns P. Therefore we shall denote them simply by the letter T, dropping the index P. We shall term the matrix $\|\alpha_{ij}\|$ the basic probability matrix of the class of perceptrons being considered.

Similarly, the class of training sequences K which are formed with the aid of the random selection of an image at each training step, regardless of the images selected in the remaining steps, can be characterized by the probability vector $(\beta_1, \beta_2, \dots, \beta_m)$ of the class being considered. For any $i = 1, 2, \dots, m$ the ith component β_i of this vector is equal to the probability of the selection of the ith vector as the image being shown at any given step of the training. In this case the ith component v_i of the characteristic vector of the class K is the number of occurrences of the event having the probability β_i with N independent trials, where N is the length of the training sequence of the class K (according to the definition of the class of training sequences, all the sequences occurring in the class have the same length).

With sufficiently large values of N, for any image i the quantity v_i can be considered normally distributed and its mathemetical expectation and variance are given by the equations

$$E(v_i) = N\beta_i; \quad D(v_i) = N\beta_i(1-\beta_i) \quad (i = 1, 2, \ldots, m). \tag{86}$$

We note that the random quantities v_i, and also the random quantities T^D_{ij}, are not, generally speaking, independent for various values of i and j, which creates additional difficulties in the calculation of the probability of correct operation of the perceptron using equations (81) and (84). However, in many cases we can avoid these difficulties by the introduction of certain additional propositions. We shall demonstrate this situation using several examples.

Example 1. We consider the discrete perceptron A with neurons of the (1, 1, 1,) type, having a regular square $(n \times n)$-retina and 2n images, which are chosen to be n horizontal lines of length n combined into the pattern P, and n vertical lines of length n combined into the pattern Q. All the images have the same probability (equal to 1/2n) of appearing in the training sequence. We assume that the perceptron A is complete. This means that in both the neuron set of the Pth pattern and in the neuron set of the Qth pattern for any method of connection of the neuron to the retina there is precisely one neuron having exactly the same connection with the retina. In the perceptron A there operates α-law encouragement with the constants a and b and the initial weights of the neurons are equal to zero.

We are required to find the training effectiveness of the perceptron A in the class of random training sequences of length 2N containing precisely N showings of the images of the first image pattern and N showings of the images of the second pattern.

Solution. The perceptron will obviously be summetrical and will therefore be completely characterized by its characteristic matrix

$\|T_{ij}\|$. It is easy to see that the neuron is stimulated by the ith image (vertical or horizontal line) if and only if its stimulating input is connected to the receptor lying on the corresponding line and its inhibiting input is connected to the receptor lying away from this line. For any given i there are in all $n(n^2 - n) = n^2(n - 1)$ different connections of this sort. In view of the completeness of the perceptron A, the following equation is valid [formula (87)]

$$T_{ii} = n^2(n-1) \quad (i = 1, 2, \ldots, 2n). \tag{87}$$

Let us assume that the numbers from 1 to n designate the horizontal lines (images of the pattern P) and the numbers from n + 1 to 2n designate the vertical lines (images of the pattern Q). By analogy with the way the expression for T_{ii} was found, we find two more expressions

$$T_{ij} = 0, \tag{88}$$

if i and j are images of the same pattern;

$$T_{ij} = (n-1)^2, \tag{89}$$

if i and j are images of different patters.

Using E to denote a unit matrix of nth order and D to denote a square matrix of order n, all the elements of which are equal to unity, we represent the characteristic matrix M of the perceptron being considered in the form

$$M = \begin{Vmatrix} n^2(n-1)E & (n-1)^2 D \\ (n-1)^2 D & n^2(n-1)E \end{Vmatrix}. \tag{90}$$

Let $(v_1, v_2, \ldots, v_{2n})$ be the characteristic vector of the class of training sequences being considered. As a result of the assumed condition, the components of this vector satisfy the condition

$$v_1 + v_2 + \ldots + v_n = v_{n+1} + v_{n+2} + \ldots + v_{2n} = N. \tag{91}$$

As the result of theorem 3, we write the necessary and sufficient condition for correct recognition of any given image i

$$\sum_{i \in P_i} T_{ij} v_j > \sum_{i \bar{\in} P_i} T_{ij} v_j \tag{92}$$

or, taking account of relations (87)-(89) and (91),

$$n^2(n-1)v_i > (n-1)^2 N. \tag{93}$$

We write relation (93) in the equivalent form

$$v_i > \left(\frac{1}{n} - \frac{1}{n^2}\right) N. \tag{94}$$

The probability of the appearance of the ith image in each of the N showings of the representations of the pattern P_1 is equal to $1/n$. Therefore for the mathematical expectation and the variance of the quantity v_i we obtain the expressions

$$E(v_i) = \frac{1}{n} N; \quad D(v_i) = N\frac{1}{n}\left(1 - \frac{1}{n}\right). \tag{95}$$

In view of theorem 3 from §3 of the present chapter, with sufficiently large N the probability q_i of satisfaction of inequality (94) can be calculated from the equation

$$q_i \approx 0{,}5 + \frac{1}{\sqrt{2\pi}} \int_0^k e^{-\frac{z^2}{2}} dz \quad (i = 1, 2, \ldots, 2n), \tag{96}$$

where k is the value of the ration of the modulus of the difference of the right side of inequality (94) and the mathematical expectation $E(v_i)$ to the mean square deviation of the quantity v_i, equal to the square root of the variance. In other words,

$$k = \frac{1}{n^2} N : \sqrt{N \cdot \frac{1}{n}\left(1 - \frac{1}{n}\right)} = \sqrt{\frac{N}{n^3\left(1 - \frac{1}{n}\right)}} \approx \sqrt{\frac{N}{n^3}} \tag{97}$$

Since the value of q_i does not depend on i, it coincides with the probability q of correct recognition by the perceptron A of any randomly selected image after the preliminary showing of the randomly selected training sequence of length 2N from the class of sequences being considered.

The value of the probability q is just the value of the overall effectiveness of the training of the perceptron A in the given conditions. We present the table of the values of the probability q for several values of k:

N	n^3	$4n^3$	$9n^3$
k	1	2	3
q	0,841	0.977	0,999

Thus, in order to reduce the probability of error of the perceptron being considered to 0.1% with random selection of the training sequence it is necessary to make use of sequences of very great length, equal to 18 n^3. At the same time, we see immediately from inequality (92) that we can reduce this probability to zero (obtaining absolutely accurate recognition) as the result of showing each image exactly one time, i.e., with the use of a sequence having a length of only 2n. This example gives a striking demonstration of the inadvisability of the use of random training sequences. At the same time it indicates the serious differences of the learning mechanism described from the learning mechanism realized in the human brain.

Actually, the latter mechanism has a marked capability for extrapolation of experience, i.e., for correct recognition of images which never appeared in the training process. At the same time the perceptron described in the example considered does not give a final guarantee of correct image recognition (with random organization of the training process) even when the average number of displays if each image reaches a very large number (of the order of n^3).

This conclusion is associated, or course, to a certain degree with the specific nature of the example. However, it is not difficult to note that with purely random connection of the (1, 1, 1)-neurons to the retina (excluding the connection of both inputs of the neuron to the same receptro) the mathematical expectation of the components of the characteristic matrix will differ from the components of the characteristic matrix of the complete perceptron only by a constant factor which is not significant from the point of view of the calculation of the training effectiveness. Therefore, with the random connection of the neurons to the retina the most probable behavior of the resulting perceptrons will be precisely that of the complete perceptron described above.

Thus, the random organization of the connections of the neurons with the retina cannot, generally speaking, provide a high quality of perceptron functioning. From theorem 3 it follows that the capability of the perceptron for extrapolation of experience is increased with increase of those components of the characteristic matrix whose indices belong to the same pattern, and with reduction of those components whose indices belong to different patterns.

We shall say that a perceptron has <u>absolute capability for extrapolation</u> if for any pattern P and any image $\underline{i}$ from this pattern training by any sequence containing the image $\underline{i}$ not less than one time will lead to a correct recognition of all the images of this pattern. We obtain the following result.

<u>Theorem 4</u>. In order that a discrete symmetric perceptron with symmetric initial conditions in which (generalized) α-law encouragement operates have absolute capability for extrapolation it is necessary and sufficient that all the components T_{ij} of the characteristic matrix of the perceptron whose indices belong to the same pattern be nonzero, and that all the components T_{ij} whose indices belong to different patterns be equal to zero.

Actually, let us assume that the condition of the theorem is satisfied. Then inequality (84) will be valid, if for any one of the image $\underline{j}$ from P_1 the value of v_j is nonzero. As the result of theorem 3, this means that the perceptron being considered has absolute capability for extrapolation.

Let us assume that the condition of the theorem is not satisfied. This leads to the consideration of two cases: 1) for some pattern Q there is a pair of images i, j belonging to it such that $T_{ij} = 0$; 2) there is a pair of images k, r belonging to different patterns and such that $T_{kr} \neq 0$. In the first case, as a result of theorem 3 the learning sequence compose exclusively from the images j does not lead to correct recognition of the image i. In the second case, let us consider the learning sequence composed of one image k and any number v_r larger than T_{kk}/T_{kr} of images r, Then, in application to the recognition of the image k the substitution of the indicated values in inquality (84) leads to the inequality $T_{kk} > T_{kr}\ v_r$. In view of the selection of v_r this inequality is not valid, which as a result of theorem 3 means the impossibility of correct recognition of the image k. Consequently, in both cases the perceptron will not have absolute capability for extrapolation, q.e.d.

Usually the images belonging to the same pattern are numbered using sequential whole numbers. In this case it is natural to partition the characteristic matrices of the symmetric perceptrons into cells corresponding to the different patterns. Absolute capability for extrapolation is achieved in this case when these matrices are cellulary diagonal and the diagonal cells do not contain zero elements. This form the cahracteristic matrices is not always completely achievable, however any good approximation to it will require, as a rule, avoidance of the completely random connection of the neurons with the retina. The effect obtained as a result of this deviation from random connection is best demonstrated using an example.

Example 2. Find the training effectiveness of the perceptron B, differing from the perceptron A of example 1 only in that it retains only those neurons, both ends of which are connected to the receptors lying either on one horizontal or on one vertical line. The training conditions are the same as in example 1.

Solution. The perceptron B, just as the perceptron A, will obviously be symmeteical. It is not difficult to find that the elements of its characteristic matrix are given by the relations $T_{ii} = n(n - 1)$; $T_{ij} = 0$ ($i, j = 1, 2, \ldots, 2n$; $i \neq j$). The condition of correct recognition of the ith image is expressed by the condition $T_{ii}v_i > 0$ or, what is the same, $v_i > 0$. In other words, for the correct recognition of the ith image it is necessary and sufficient that it was shown at least once to the perceptron in the process of its training.

With N random displays of the images of one pattern, the probability of the nonappearance in the training sequence of the ith image is obviously equal to $\left(1-\frac{1}{n}\right)^N \approx e^{-\frac{N}{n}}$ and the overall effectiveness of the training is expressed by the number $1-e^{-\frac{N}{n}}$. In order to reduce the probability of incorrect operation of the perceptron to 0.1%, as was done in example 1, it is sufficient to set $N = 7n$, or, in other words, to use a training sequence of length $14n$. We recall that in the first esample the same training effectiveness was obtained only by using a training sequence of length $18n^3$.

It is curious that such a sharp increase of the training effectiveness is obtained not as a result of more complication, but as a result of the simplication of the perceptron, since the perceptron B is obtained from perceptron A by discarding a large number of neurons poorly connected to the retina. It is easy to find that the total number of neurons in the perceptron A is $2n^2(n^2 - 1)$ while that in perceptron B is only $4n(n - 1)$. This situation once again indicates the imperfection of the perceptron learning mechanism and its significant difference from the learning process which take place in the human brain.

Let us consider another exampoe of the computation of the learning effectiveness in a class of perceptrons.

Example 3. Determine the training effectiveness of the class C of discrete symmetric perceptrons with symmetric initial conditions subject to generalized α-law encouragement. The retina, patterns and images are the same as in example 1. The number of neurons of each of the two existing patterns is equal to N. The inputs of all the neurons are connected independently of one another with equal probability to any receptor of the retina, excluding only the case of simultaneous connection of both inputs of a neuron to the same receptor. The training sequence contains each of the 2n images exactly once each.

Solution. It is easy to see that the components T_{ij} of the characteristic matrix of the class C, in which the indices i and j are different images of the same pattern, are equal to zero. The condition for correct recognition of the ith image, given by theorem 3, is written in our case

$$T_{ii} > \sum_{l \in P_1} T_{il} \qquad (98)$$

It is easy to see that the set M_{ij} of neurons of the same pattern which are stimulated by both the ith and the jth image with different j, differing from i, are disjoint. All these sets are contained, of course, in the set M_{ii}. Since T_{ij} is just the number of elements of the set M_{ij}, for the satisfaction of inequality (98) it is necessary and sufficient that among the neurons of the pattern P_1 there be at least one neuron which is stimulated by the ith image but is not stimulated by any image of the opposite (different from P_1) pattern.

From the geometry of the images it follows directly that this condition is satisfied by the neurons both of whose inputs are connected to the same vertical (if i is a horizontal line) or to the same horizontal (if $\underline{i}$ is a vertical line). For any fixed $\underline{i}$ from the total number of different connections this condition is satisfied only by $n^2(n^2 - 1)$ connections. The probability of the desired connection is therefore equal to $n(n - 1)/n^2(n^2 - 1) = 1/n(n + 1)$ and the probability that such a connection will not take place for any of the N neurons N neurons is equal to $\left(1-\frac{1}{n(n+1)}\right)^N \approx e^{-\frac{N}{n(n+1)}}$. Consequently, the overall training effectiveness is expressed by the equation

$$r \approx 1 - e^{-\frac{N}{n(n+1)}}.$$

If the number of neurons of each pattern is equal to $7n(n + 1)$, i.e., exceeds by approximately a factor of 7 the total number of receptors, then the probability of incorrect operation of a perceptron randomly selected from the class C after training by display of all the images one time each will be equal to e^{-7}, which is equal to about 0.001.

As mentioned above, the construction of the theory of perceptron learning indicates the basic differences of the learning process realized by it from the actual learning process of the human brain. Changing from the discrete neurons to the continuous, or replacing the α-law encouragement by β- or γ-law does not significantly alter this situation. The situation may be rectified partially by the addition to the processes realized in the perceptron of reconnection of the neurons which interfere with or do not significantly aid the learning process.

We can provide for, for example, periodic verification of the weights of the neurons and random connections of the neurons with smaller weight. Mechanisms of this sort are realized in Roberts' adapt [67] and the Selfridge pandemonium [72]. They increase the equipment utilization coefficient and reduce the number of neurons, which in the perceptron schemes with purely random connections reach tremendously large values.

However, this is completely inadequate for clarification of such a feature of the adaptive functions of the brain as the use of particular features distinguished on patterns already studied for the acceleration of the process of learning to recognize new patterns containing all or part of these features. It is easy to see that such a process can be realized in the multi-stage perceptrons, i.e., in those circuits in which the pattern summators of the perceptron of the lower stage are used as the repectors for the perceptron of the following stage. Here the perceptrons of the lower stages are taught to recognize individual properties of the patterns and the perceptrons of the higher stages are trained to recognize the ensembles of thes properties. Corresponding alterations and complications of the laws of encouragement can be accomplished in many different ways. We note that the scheme which essentially includes the idea of the two-stage perceptron is used in the algorithm for teaching the recognition of geometric figures described in the work of Glushkov, Kovalevskiy and Rybak [29].

Introduction of these improvements still does not permit approaching the simulation of another important characteristic of the brain, that is, the establishment of the invariance of all the patterns with respect to their movement and to change of dimensions on the basis of a <u>limited experience</u>, using only a <u>small part</u> of all the patterns. To achieve any success in this direction we must alter not only the construction of the perceptron but also the very methodology of the learning process. To do this we introduce the possibility of the recognizing device itself participating in the organization of the learning sequence.

If, for example, the recognizing device A is shown as representatives of a particular pattern several different images, then the device A must have the possibility of repeating the demonstration of these images as many times as necessary to ensure their correct recognition in the future. Moreover, the device must have the possibility of repeating the display of those same images subjected to those variations which the image of an object on the retina of the eye is usually subject to with changes of the relative position of the eye and the object being considered.

We can, of course do things other than introduce the described feedback which permits the recognizing device to alter the learning sequence. In place of this, the recognizing devices themeselves can be constructed so that after the display of a particular image there is an increase of the probability of the display at the following step of the same image, viewed, perhaps, at a different angle, or at least images belonging to the same pattern. In other words, in the training of the recognizing devices we must avoid the construction of the learning process using the scheme of independent trials and go to the more complex schemes described by the Markov chains.

The suggested variations of the methods of construction of the learning sequences considerably improve the functioning of the recognition devices in the simple learning regime. However, it is in the self-learning regime that these variations are of principal importance, since it is only in this direction that we can hope that the classification of images performed by the self-training devices will correspond to the original classification performed by a human. It is clear that the description of processes of this sort requires far more complex mathematical apparatus than that which has been used in the present section.

§7. OPERATION OF THE DISCRETE α-PERCEPTRONS IN THE SELF-LEARNING REGIME

In the preceding section we studied the behavior of the discrete α-perceptrons in the learning regime. The characteristic feature of the learning regime is the presence of the teacher, who knows the correct classification of the images. In the present section we shall study some questions associated with the behavior of the discrete α-perceptrons in the self-learning regime. In this case the teacher is missing, and the processes of self-organization which lead to the alteration of the image classification performed by the perceptron are determined by the positive feedback introduced into the perceptron circuit.

It is well known that the analysis of the behavior of the perceptrons in the self-learning regime which was made by Rosenblatt [69] is very far from being mathematically rigorous. The absence of rigorously proved propositions in this field leads some authors to ascribe to perceptron self-learning (particularly in publications of a popular science nature) many properties which in actuality it does not possess and cannot possess. On the basis of the considerations of the present section, it is not difficult to draw several conclusions which outline the boundaries of the actual possibilities inherent in the self-learning of the perceptrons.

Let us consider the discrete α-perceptron designed for the recognition of the two patterns P and Q. As the single output signal of the perceptron we shall consider the difference of the signals of the summators of the Pth and Qth patterns

$$V_i(l) = U_i^P(l) - U_i^Q(l). \qquad (99)$$

Here, just as in the equations of the preceding section, the index i runs through all the images of the (both Pth and Qth) patterns, l is any sequence of images shown to the perceptron in the process of its self-training.

Just as in the case of training, it is not difficult to show that the functioning of the symmetric perceptron in the self-training regime is determined by the sum a + b of the encouragement and penalty constants, and not by these constants considered separately. Having in view also the possibility of arbitrarily varying the scales, it is permissible, without losing generality, to assume that a = 1, and b = 0. In the future we shall always make this assumption.

Using $\|T_{ij}\|$ to denote the characteristic matrix of the perceptron and recalling the definition of α-law encouragement, we easily obtain the equation

$$V_i(lj) = V_i(l) + T_{ij} \operatorname{sign} V_j(l). \qquad (100)$$

Here the symbols lj denote the image sequence l to which there is appended the image j.

Equation (100) is valid for any pair of images i, j and for any image sequence l. The function sign x, as usual, is taken equal to plus 1 for positive values of x and equal to minus 1 for negative values of x. It is clear that in the case of a zero value of the quantity $V_j(l)$, from the exact meaning of the encouragement law (positive feedback) the quantity sign $V_j(l)$ in equation (100) must be undefined. In order to avoid indefiniteness, in the future we shall, by definition, consider zero to be a positive quantity, so that sign 0 = + 1.

Keeping in mind the indicated modification in the definition of the function sign x, we shall consider equation (100) as a method of recurrent specification of the vector $V(l) = (V_1(l), V_2(l), \ldots, V_m(l))$, which defines the output signals of the perceptron under the action of any image $j = 1, 2, \ldots, m$ after the application to the perceptron input of the image sequence l. The initial value of this vector $V(0) = (V_1(0), V_2(0), \ldots, V_m)(0))$ is assumed given. The image is associated by the perceptron with the P pattern or the Q pattern in accordance with whether or not the corresponding component $V_j(\ell)$ of the vector being considered is positive or negative (we recall that zero, according to the accepted agreement, is considered to be a positive number).

Since all the quantities T_{ij} are integral (and also nonegative) numbers, the problem of the design of the perceptron in the self-training regime reduces in essence to the problem of a random walk over a discrete lattice in space with the number of measurements $\leq m$. Assuming that the display of the images in the self-learning process is performed following the scheme of independent trials, it is easy to note that the probabilities of the transitions from any point of such a lattice are determined only by the ensemble of signs of the coordinates of this point.

It is not difficult to see that the set of signs of the corrdinates of any point of the lattice also defines the image classification performed by the perceptron which has as the vector of its output signals the radius-vector of this point. From the point of view of perceptron theory, of prime interest is the limit distribution of the signs of the coordinates of the vector $V(\ell)$ with unlimited increase of the length of the training sequence ℓ. The analysis made above shows that the required distribution is obtained from the limit distribution for the Markov chain corresponding to the walk over the discrete lattice described above.

Since this chain has an infinite number of states, finding the distribution limit in the general case is quite complex. We can, however, note several cases when finding the limit distribution is easily reduced to the study of the Markov chain with a finite number of states.

Let us consider as an example the discrete symmetric α-perceptron

A designed for the recognition of 2n images, the first n of which belong to the pattern P, and the last n to the pattern Q. Assume further that for the elements of the characteristic matrix of the perceptron A the relation $T_{ij} = a > 0$, holds if i and j belong to the same pattern and $T_{ij} = 0$, holds if i and j belong to different patterns. In view of theorem 4 from §6 of the present chapter, the perceptron being considered has absolute capacity for extrapolation and, consequently, behaves itself best in the learning regime (learns the correct recognition as a result of showing at least one image of each pattern). Let us assume that the initial conditions will be the conditions $V_i(0) = b$ $(b > 0)$ for $i = 1, 2, \ldots, m$ $(m \leq n)$ and $V_j(0) = -b$ for $j = m + 1, m + 2, \ldots, n, \ldots, 2n$.

From equation (100) it follows directly that $V_j(\ell) < 0$ for any sequence ℓ with $j = n + 1, n + 2, \ldots, 2n$. The remaining components will be expressed by the equations $V_i(\ell) = b + ka$ for $i = 1, 2, \ldots, m$ and by $V_i(\ell) = - b + ka$ for $i = m + 1, m + 2, \ldots, n$, where k is the difference between the number of appearances of the images corresponding to the positive components $V_i(\ell')$ and the number of images corresponding to the negative components $V_i(\ell')$ (ℓ' is the corresponding subsequence of the sequence ℓ).

Let us assume that the self-training process is accomplished using the scheme of independent trials with different probabilities of the appearance of all the images. Since the display of the images of one pattern in the case considered has no effect on the recognition of the images of the second pattern, we can without losing generality assume that in the self-training process there participate only the images of the pattern P (the images of the pattern Q always correspond to a negative output signal regardless of whether they are included in the self-training process or not).

Let us assume that b does not contain a, and use t to denote the whole number $[a/b] + 1$. It is not difficult to see that for the study of the functioning of the perceptron A only those values of the parameter k are of interset which are included in the closed interval $[-t, t]$. Actually, if in the self-training process the quantity k reaches the value t even one time, then in the future, as a result of equation (100), it can only increase, and the perceptron, beginning with that moment, will deliver a positve output signal for all images of the pattern (which corresponds to correct classification). Similarly, if the parameter k takes the value $-t$, the perception will deliver a negative output signal for all images (which actually means the absence of any image classification, since all the images are associated by the perceptron to the same pattern).

Now, as is easily seen, the limit behavior of the perceptron A is determined by the Markov chain with $2t + 1$ states $k = -t, -t + 1, \ldots, \ldots, -1, 0, 1, \ldots, t-1, t$. Inview of the assumption made on the probabilities of the appearance of the images in the process of the self-training, for any k differing from t or $-t$ the probability of transition into the state $k + 1$ is equal to m/n, and the probability of transition into the state $k - 1$ is equal to $n - m/n$. From the state t (just as from the state $-t$) transition is possible only into the same state, since from the point of view of the functioning of the perceptron all states with $k > t$ (correspondingly – with $k < -t$) do not differ from the state $k = t$ (correspondingly – from the state $k = t$).

Introducing the notations $p = m/n$ and $q = n - m/n$, we obtain for the considered Markov chain the matrix of the transition probabilities

$$M = \begin{Vmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ p & 0 & q & 0 & \cdots & 0 & 0 & 0 \\ 0 & p & 0 & q & \cdots & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdots & 0 & q & 0 \\ 0 & 0 & 0 & 0 & \cdots & p & 0 & q \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{Vmatrix}$$

This matrix has unity as its double characteristic root. The probabilities of the transition of the chain into the states <u>t</u> and $-t$ are equal to the limit tranistion probabilities $p^{\infty}_{t+1,1}$ and $p^{\infty}_{t+1,2t+1}$. For the probability $p^{\infty}_{t+1,i}$ we obtain from the Perron equation

$$p^{\infty}_{t+1,i} = \lim_{s\to\infty} \frac{d}{d\lambda} \frac{\lambda^s M_{i,t+1}(\lambda)}{\psi_1(\lambda)}\Bigg|_{\lambda=1}. \qquad (101)$$

It is easy to see that $M_{i,t+1}(\lambda)$ for <u>i</u> differing from 1 and from $2t + 1$ contains $(\lambda - 1)^2$ and therefore for all values of <u>i</u> $p^{\infty}_{t+1,i} = 0$. For $i = 1$ $M_{1,t+1}(\lambda) = (\lambda - 1) M_1(\lambda)$ and for $i = 2t + 1$ $M_{2t+1,t+1}(\lambda) =$ $= (\lambda - 1) M_2(\lambda)$, where $M_1(\lambda) = p^t Q(\lambda)$, $M_2(\lambda) = q^t R(\lambda)$.

From equation (101) we easily obtain $p^{\infty}_{t+1,t} = cp^t$, $p^{\infty}_{t+1,2t+1} = cq^t$.

Since all the remaining limit transition probabilities in the $(t + 1)$<u>th</u> row are equal to zero, from the conditions of stochasticity of the mateix of the limit transition probabilities we find the value of <u>c</u>: $c = 1/p^t+q^t$. Thereby we have proved the following proposition.

With unlimited continuation of the self-training process the perceptron A described above with the probability p^t/p^t+q^t establishes the correct classification of the images and with the probability q^t/p^t+q^t relates all the images to the same pattern.

The considered example, as the attentive reader can easily note, strictly speaking cannot be performed in a real perceptron, except for the trivial cases $m = n$, $p = 1$, $q = 0$ and $m = 0$, $p = 0$, $q = 1$. The reason is that with the assumptions made relative to the characteristic matrix all the images of the same pattern stimulate the same set of neurons. Therefore, the output signals induced by the images of the same pattern always must be eq ıl to one another, including at the initial moment.

It is not difficult, however, to note that by setting $T_{ii} = a+\delta$ for all $i = 1, 2, \ldots, 2n$ $(\delta > 0)$, we obtain the possibility of satisfying the initial conditions introduced in the example. Moreover, if δ is significantly smaller than a, and t is relatively large, then the perceptron behavior described in the example can serve as a good approximation for its real behavior.

Let us consider the complete discrete α-perceptron B with (1, 1, 1) -neurons, with a square $(n \times n)$-retina, designed for the recognition of the two patterns P and Q. The pattern P consists of n horizontal lines, and the pattern Q and n vertical lines. Each of these lines constitutes an individual image. In the preceding section it was noted that the perceptron B ı ,uld be considered as the most characteristic representative of the class of perceptrons with random connections of the neurons with the retina. According to theorem 1 and the corollary following it from the work of Rosenblatt [69], such perceptrons constructed using continuous neurons with self-learning must tend to a state in which all the images are related to the same pattern with a probability arbitrarily close to unity. Let us show that this statement is not valid for the perceptron B.

It is easy to see that we can select any initial conditions for the perceptron B. We shall term the smallest of the numbers $|V\ (0)|$ $(i = 1, 2, \ldots, 2n)$ the lower boundary of the moduli of the initial conditions. With the assumptions made, the following theorem is valid.

Theorem 1. For any arbitrarily small positive number ε there is a number S such that in the case when the lower boundary of the moduli of the initial conditions exceeds S the perceptron B in the self-training regime (with equiprobability of appearance of all the images) retains the intial classification of the images with the probability $p > 1 - \varepsilon$.

Proof. We use N to denote the length of the training sequence ℓ and v_i to denote the number of appearances of the ith image ($i = 1, 2, \ldots, 2n$) in this sequence. Let $V_i(0) = x_i$ ($i = 1, 2, \ldots, 2n$); k_i is the set of all indices j (images) relating to the pattern opposite in comparison with i and such that the sign of x_j coincides with the sign of x_i; z_i is the set of all indices j relating to the pattern which is opposite to i and such that the sign of x_j is opposite to the sign of x_i (as before, zero is here considered to be a positive number).

As was shown in the preceding section, the arbitrary element T_{ij} of the characteristic matrix of the perceptoon B is equal to $n^2(n-1)$, 0 or $(n-1)^2$ depending on whether the indices i and j coincide or do not coincide but relate to the same pattern, or do not coincide and relate to different patterns. Using this circumstance, with the aid of equation (100) we easily learn that the original classification of the images is retained in the self-training process if for all N = 1, 2, ... the following inequalities are satisfied

$$n^2(n-1)v_i + (n-1)^2\Big(\sum_{j\in k_i} v_j - \sum_{j\in z_i} v_j\Big) + |x_i| > 0 \quad (i = 1, 2, \ldots, 2n),$$

and even more so if

$$n^2(n-1)v_i - (n-1)^2 \sum_{j\in k_j \cup z_j} v_j + x > 0 \quad (i = 1, 2, \ldots, 2n), \qquad (102)$$

where x is the minimal of the numbers $|x_i|$ ($i = 1, 2, \ldots, 2n$). In turn it is not difficult to verify that inequalities (102) are satisfied if the inequalities

$$\left|\frac{v_i}{N} - \frac{1}{2n}\right| < \frac{1}{4n^2}\left(1 + \frac{2x}{N(n-1)}\right) \quad (i = 1, 2, \ldots, 2n). \qquad (103)$$

are satisfied.

With designation of the quantity $1/4n^2$ by the letter δ it is evident that the inequalities (103) will be obviously satisfied if the inequalities

$$\left|\frac{\nu_i}{N}-\frac{1}{2n}\right|<\delta \qquad (i=1, 2, \ldots, 2n). \tag{104}$$

are satisfied.

As the result of theorem 4 of §3 of the present chapter, there exist the positive constants a and b, not depending on N, such that the probability P_N of nonsatisfaction of at least one of the inequalities (104) with any fixed value of N has the estimate $R(M)=\frac{a}{M^{n-\frac{1}{2}}}\cdot\frac{e^{-bM}}{1-e^{-b}}$. The probability of nonsatisfaction of at least one of inequalities (104) for values of N from M to ∞ does not exceed the sum of the series $\sum_{N=M}^{\infty}\frac{a}{N^{n-\frac{1}{2}}}e^{-bN}$, , and this sum is clearly less than $P_N<\frac{a}{N^{n-\frac{1}{2}}}e^{-bN}$. With $M\to\infty$ the quantity R(M) vanishes. Let us take M so that $R(M)<\varepsilon$.

Now taking $S = 2(M-1)n^2(n-1)$ we find that with $x > S$ the inequalities (103) are satisfied foe all values of $N = 1, 2, \ldots, M-1$. As a result of the choice of M, for all the remaining values of N the inequalities (103) are satisfied with a probability greater than $1-\varepsilon$. Since satisfaction of the inequalities (104) for all values of N from 1 to $<\infty$ means retention of the original classification of the images, the theorem is proved.

Theorem 1 shows that with sufficiently large initial values of the output signals for all the images the considered perceptron actually is practically devoid of capability not only for self-training, but even to simply change the image classification initially specified to it. From the proof of the theorem it is easy to see that the remaining weak capability for self-alteration has its maximal value in the case of the correct initial classification. In other words, the perceptron has the least tendency to retain the correct method of functioning.

Let us consider β-law encouragement. To do this let us fix the arbitrary number β included between zero and unity, and let us consider the arbitrary symmetric perceptron C with β-law encouragement whose characteristic matrix is diagonal, i.e., in other words, has nonzero elements only on the principal diagonal. As shown in the preceding section, this property is possessed by the summetric perceptron C_1 with (1, 1, 1)-neurons which is designed for the recognition of horizontal and vertical lines and in which the inputs of each neuron are connected to the elements of the retina located on one horizontal or on one vertical.

In the case of β-law encouragement the basic recurrent relation for the determination of the output signals is written

$$V_i(lj) = (1-\beta)(V_i(l) + T_{ij}\,\mathrm{sign}\, V_j(l)). \qquad (105)$$

The notations here are exactly the same as in equation (100) and this relation is also valid for any discrete symmetric perceptrons. In the case of perceptrons with diagonal characteristic matrix both terms in the right side of equations (100) and (105) always have the same sign (the case when the second term is equal to zero is excluded from consideration). Whence follows directly the validity of the following proposition.

<u>Theorem 2</u>. The discrete symmetric perceptron C with diagonal characteristic matrix is completely devoid of capability for self-learning (i.e., retains unchanged any given original image classification) both in the case of α-law encouragement, and in the case of β-law encouragement.

It is also easy to see the validity of the following proposition.

Theorem 3. No discrete symmetric perceptron (with either α- or β-law encouragement) operating in the self-training regime can alter the original image classification if this classification relates all the images to the same pattern.

Our results can be considered as counterexamples to the results of Rosenblatt [69], to the degree that his discussions relate not only to the continuous but also to the discrete neurons. In any case these results indicate that the asymptotic behavior of the perceptrons in the self-training regime is far more complex and requires considerably more precise techniques for its study in comparison with the techniques of purely qualitative nature used by Rosenblatt [69].

For a visual representation picture of the peculiarities of the behavior of the perceptrons in the self-learning regime in comparison with the learning regime, let us consider the case when the number of images is equal to two (each pattern consists of one single image). This case permits simple graphical interpretation.

First we note that in the case of the presence of two patterns (but with an arbitrary number of images) the functioning of the perceptron in both the learning and the self-learning regimes is conveniently characterized by a vector with the components $V_i(\ell)$ (see above). The basic recurrence relation for these components will obviously have the form

$$V_i(lj) = V_i(l) \pm T_{ij}. \qquad (106)$$

This relation is valid for any pair of images i, j and for any training sequence ℓ. The second term in the right side is taken with the plus sign if the image j in the correct classification relates to the positive output signal, and with the minus sign if the corresponding output signal must be negative.

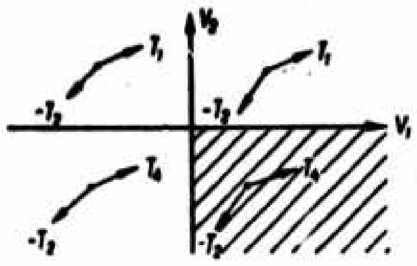

Fig. 13

Let us consider the discrete symmetric perceptron with α-law encouragement whose characteristic matrix is the matrix $T = \begin{Vmatrix} a & b \\ b & a \end{Vmatrix}$, where $a > b > 0$. Let us assume that with correct classification the firs image must induce a positive output signal, and the second image – a negative output signal. Plotting the coordinate $V_1(\ell)$ along the horizontal axis, and the coordinate $V_2(\ell)$ along the vertical axis, we associate with each vector $(V_1(\ell), V_2(\ell))$ some point of the plane. Selecting one point in each quadrant, we obtain a visual impression of the action of equation (106) (Fig. 13).

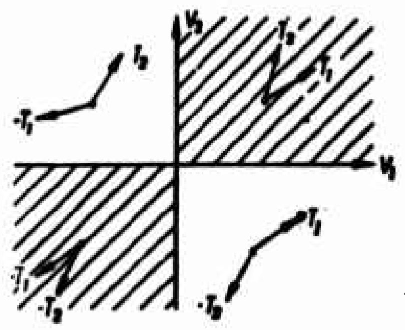

Fig. 14

In Fig. 13 the letter T_1 denotes the vector (a, b) and the letter T_2 denotes the vector (b, a); the characteristic feature of the training regime is that the directions of the vectors (defining the random walks of the point on the lattice) do not depend on the position of the points on the plane. The resultant of these vectors is always directed in the direction of that quadrant in which the signs of the coordinates (output signals of the perceptron) coincide with the correct classification (in the present case this quadrant is the hatched – fourth quadrant).

For the case of the self-training regime the interpretation of the corresponding equation (100) is shown in Fig. 14. In contrast with the previous case, the directions of the vectors which define the random walks are different in the different quadrants. The designations of the vectors are the same as in Fig. 13.

It is not difficult to note the qualitative differences of the situation shown in Fig. 14 from the situation shown in Fig. 13. First of all, the first and third quadrants (shown shaded in Fig. 14) now possess a trapping property: a point which falls into one of these quadrants in the process of the random walk can never excape from it.

Entrance into these quadrants means actually the absence of any classification (both images are assigned to the same pattern). Moreover, from the quadrant corresponding to correct classification with accuracy to the sign of the output signal (third quadrant) there is always a zero probability of exit into the neighboring quadrants.

Considering the resulting situation in the purely qualitative aspect, similar to the approach of Rosenblatt [69], we would have to come to the conclusion that the perceptron which we are studying tends asymptotically to a state in which output signals of the same sign (absence of any classification) are generated for all images. A more thorough consideration (repeating the analysis made in the proof of theorem 1) leads, however, to a completely different conclusion: just as in the case of theorem 1, with sufficient removal of the initial point from the boundaries of the quadrant the probability of continuation of the random walk without leaving this quadrant in all the succeeding instants of time (clear up to infinity) can be made arbitrarily close to unity.

This once again underscores the danger arising in the case when general conclusions on the asymptotic behavior of the perceptrons are based on arguments of purely qualitative character, without confirming them by exact computations and estimations.

§8. LOGICAL CLASSIFICATION SYSTEMS AND CONDITIONAL PROBABILITY MACHINES

The systems for pattern recognition considered in the preceding sections are devices for the classification of certain subsets in the image space. Directing ourselves to the visual, audible and other patterns which are continuous in nature, we to a certain degree transferred the property of continuity to the corresponding classification systems. Actually, even in the case of clearly discrete receptors the hypothesis of the N-extrapolatability of the patterns, which presumes continuity of the patterns, permitted selection for classification only of those sets which were in the corresponding sense "well arranged." This same implicit use of the property of the continuity of the patterns is also present in the perceptrons (including the discrete) and also in all the other algorithms and devices for the recognition of patterns mentioned in the preceding sections.

The limitation of the number of image space subsets which are subject to consideration and classification in the case of the visual, audible and other patterns which are of a continuous nature is of prime importance, since without such limitation the recognition problem would be practically unsolvable for these patterns.

Actually, in the case when the retina consists of n binary receptors the image space consists of $I(n) = 2^n$ different images and contains $Q(n) = 2^{2^n}$ different subsets – possible discrete patterns. With $n = 5$ the second of these quantities already exceeds four billion, and with a relatively small number of receptors such as 100 the first quantity is expressed by one with thirty zeros.

In view of these discussions it becomes evident that the problem of the construction of devices which store (or generate) the features of all possible patterns for any large values of n is practically unsolvable. However, in the case when the number of (binary) receptors does not exceed ten or fifteen it is in practice quite possible to construct a machine capable of remembering and performing various operations with all the images (but not with all the patterns) which can be reproduced with the aid of the corresponding retina.

The machines which classify all the possible images which can be obtained from binary receptors will be termed logical classification machines. We shall describe one of the possible schemes of such machines proposed by Attlee [2].

For each property of the image the Attlee classification machine contains the so-called discriminative element which is stimulated under the action of this property. Here and hereafter, by image property we shall mean the presence in some set M of receptors (depending on the choice of the corresponding property) of a definite combination of output signals (zeros and ones). Here the receptors which do not appear in the set M can have any output signals. The property that the receptors with the numbers $i_1, i_2, \ldots, i_m$ have unity output signal and the receptors with the numbers $j_1, j_2, \ldots, j_n$ have a zero output signal will be designated by $(i_1, i_2, \ldots, i_m; \bar{j}_1, \bar{j}_2, \ldots, \bar{j}_n)$.

From these definitions it becomes clear that the specification of a property is equivalent to the assignment to the retina receptor output signal of one of three values: one, zero, indifferent. If we denote the total number of receptors composing the retina by N, then it is easy to see that with a total number of images equal to 2^N, the number of their different properties will be equal to 3^N. All the properties of any given image can be obtained with the aid of replacement of some number of the signals (zeros and ones) composing this image by the indifferent signals. The number of such replacements (and this means the number of properties of each image) will clearly be equal to the sum $C_N^0 + C_N^1 + \ldots + C_N^N = 2^N$. Thus, each image causes the stimulation of 2^N elements of the classification machine.

Let us term theproperty of the image to stimulate the ith receptor the ith elementary property, and any combination of elementary properties we shall term a positive property. With N binary receptors there are in all only N elementary properties. It is also clear that the specification of any positive property is equivalent to the specification of some subset of receptors which induce unit output signals. The total number of positive properties is thus equal to the number of subsets of the set of N elements, i.e., 2^N.

Attlee terms the classification machine just described which is able to differentiate any image properties, the <u>binary classification machine</u>, in contrast with the so-called <u>unitary classification machine</u>, which is capable of differentiating only positive properties. It is easy to see that for every binary machine there exists its equivalent with respect to the classification being performed) unitary machine containing twice the number of receptors. We need only add to each receptor which reacts to a particular elementary property another receptor which reacts to the absence of this property. Although at first glance it appears that after this we need 4^N discriminative elements, in actuality many of them are redundant since they will never be stimulated. After removal of the redundant elements the number of remaining elements will be exactly the same as in the case of the binary machine, i.e., 3^N. The simplicity of the reduction of the binary machines to unitary machine makes it possible for us to limit ourselves in the future to the consideration of only the unitary machines.

It is convenient to picture the discriminative elements of the unitary classification machine with N receptors in the form of neurons having from one to N input channels and capable of being stimulated only in the case of simultaneous stimulation of all their input channels. Each of the neuron input channels is connected to the output channel of some receptor. Neurons with inputs connected to the receptors with the numbers i_1, i_2, ..., i_k, will correspond to the positive property (i_1, i_2, ..., i_k) and will be stimulated only with the presence of this property in the image being recognized. In order that the total number of neurons be equal to exactly 2^N it is necessary to assume the presence of still another neuron without input channels which is stimulated constantly regardless of the image being recognized. This neuron corresponds to the property which is the combination of the empty set of the elementary properties.

Let us consider the arbitrary receptor $\underline{i}$ and all the positive properties containing the elementary property $\underline{i}$. Among these properties there is exactly one property containing one elementary property (in the present case this will be the property $\underline{i}$ itself), exactly $C^1_{N-1} = N - 1$ properties containing two elementary properties each (all properties of the form (i, j), where $j \neq i$), exactly C^2_{N-1} properties containing three elementary properties each, etc. In the unitary machine one neuron corresponds to each of the positive properties. The total number of neurons connected to the receptor $\underline{i}$ is expressed by the sum $1 + C^1_{N-1} + C^2_{N-1} + \ldots + C^{N-1}_{N-1} = 2^{N-1}$, which amounts to exactly half of all the neurons in the unitary machine.

Making a random selection of the neurons with the condition that all the neurons are considered equiprobable, we come to the conclusion: the probability that the neuron thus selected will be connected to any given receptor $\underline{i}$ is equal to 1/2. Thus, a connection scheme which is to a certain degree close to the scheme of the unitary classification machine can be obtained as the result of the random connection of the neurons to the receptor with equal probability of connection or nonconnection of the input channels of a given neuron to a given receptor.

In the described scheme of the classification machine (either bianry or unitary) there is complete absence of any elements of self-organization or self-improvement. Therefore, following Attlee, we introduce changes and additions into the scheme of the unitary machine, after which this machine is converted into the so-called conditional probability machine.

For simplification of the notations we shall denote any positive properties of the images by capital Latin letters. If I and J are positive properties, then we use $I \cup J$ to denote the union of these properties, i.e., the positive property consisting of all the elementary properties occurring either in the property I or in the property J, or in both of these properties at the same time. We use $I \cap J$ to denote the intersection of the properties I and J, i.e., in other words, the positive property consisting of all those and only those elementary properties which occur simultaneously in both the property I and in the property J.

Let us now assume that to the input of some unitary machine there is applied some training sequence, i.e., simply speaking, some sequence of images. Generally speaking, not all the terms of this sequence possess the fixed property I. Therefore the neuron corresponding to the property I is stimulated by some terms of the training sequence and not by other terms. The ratio of the number of terms of the training sequence possessing the property I and, consequently, stimulating the indicated neuron, to the total number of terms of this sequence is naturally termed the property frequency for the given training sequence (which we shall also term the training history). For clearer differentiation from the conditioned frequency which is introduced later, we customarily term thefrequency just defined the unconditioned frequency.

We designate the unconditioned frequency of the property I by p(I); here the training history is assumed to be fixed.

Let us impose on the neurons of the unitary classification machine the additional function of computing the unconditioned frequency of the appearance of the properties corresponding to them. If the image being applied to the machine input possesses some property I, then the neuron corresponding to this property, after calculating and memorizing the unconditioned frequency of the property I, delivers at the given moment the output signal equal to one. If, however, this neuron is not stimulated (i.e., if the current image does not possess the property I), then its output signal will be the value of the unconditioned frequency of the property I which is stored in the neuron.

With the indicated additions and alterarions the unitary machine now takes on certain features which are characteristic, if not of the self-organizing automata, in any case, of the self-adaptive automata. Further improvement involves the computation of the so-called conditioned frequencies of the properties being classified by the unitary machine.

The conditioned frequency p(I/J) of the property I with relation to the property J is the ratio of the number of cases of joint appearance of the properties I and J (i.e., in other words, the appearance of the property $I \cup J$) to the total number of cases of appearance of the property J

$$p(I/J) = \frac{p(I \cup J)}{p(J)}. \qquad (107)$$

We shall as before consider that the neurons of the unitary machine compute and remember the unconditioned frequencies of the properties corresponding to them. Just as before, in the case of direct stimulation of a neuron (i.e., in the case of the presence in the current image of the property corresponding to this neuron) the neuron will deliver a signal equal to one. However, all the neurons which are not subjected to direct stimulation must now deliver not the unconditioned, but theconditioned frequencies of the properties corresponding to them. The only question is: relative to what property are the indicated conditioned frequencies to be calculated. It is easy to see that as the property J it is most natural to select the maximal positive property of the image being considered, which is the union of all the elementary properties which the given image possesses. Actually, only the very maximal property completely determines the image corresponding to it, so that the conditioned frequencies will be actually referred to the frequency of the appearance of this image.

Let us introduce the concept of subordination for the neurons of the unitary machine. We say that the neuron A is subordinate to the neuron B if the property J corresponding to the neuron B includes in itself all the elementary properties from the property I corresponding to the neuron A.

The neuron Q, corresponding to the maximal positive property of some fixed image, is obviously characterized by the subordination to it of all the neurons which are directly stimulated by this image, and it is not subordinate to any of these neurons, except, or course, itself. All the neurons to which the neuron Q is subordinate constitute the so-called superset M(Q) of this neuron. In the case being considered none of the neurons P from the set M(Q), except Q itself, is directly stimulated and therefore must deliver a signal equal to the conditioned frequency p(I/J) of the property I, corresponding to the neuron P, relative to the property J, corresponding to the neuron Q. From the definition of the superset it follows directly that $I \cup J = I$. Therefore, as the result of equation (107),

$$p(I/J) = \frac{p(I)}{p(J)}. \qquad (108)$$

Thus, for all the neurons from the superset M(Q) of the neuron Q the output signals can be determined from the equation (108): to obtain the output signal of the neuron P from M(Q) the value stored in it of the unconditioned frequency of the property corresponding to it must be diveded by the value fo the unconditioned frequency stored in the neuron Q. Attlee terms this operation for obtaining the output signals of the neurons from the superset M(Q) the <u>supercontrol operation</u>. The superconteol operation does not lead to contradiction even for the neuron Q itself, since in this case the output signal computed using equation (108) will obviously be equal to unity, which agrees with the known value of the output signal of the neuron Q obtained from the condition of the direct stimulation of this neuron.

The set of all the neurons subordinate to any given neuron P will be termed the <u>subset</u> of this neuron and will be denoted by L(P). Using the concept of the subset, it is not difficult to determine the method of obtaining the output signals for all the neurons which are not subjected at the given moment to direct stimulation and do not enter into the superset of the neuron Q corresponding to the maximal positive property of the image being considered in the given step.

Let us denote the set of all such neurons by K, and let R be any neuron from K. If as before J denotes the property corresponding to the neuron Q, I the property corresponding to the neuron R, then the neuron P which corresponds to the property $I \cup J$, will obviously belong to the superset M(Q).

As the result of equations (107) and (108), the output signal of the neuron R is equal to the output signal of the neuron P. Moreover, it is clear that the neuron R occurs in the subset L(P) of the neuron P. This suggests the conclusion that all the output signals not so far determined (for the neurons of the set K) can be obtained as the result of simple transfer of the output signals of the neurons from the superset M(Q) to the neurons of the subsets corresponding to them. It is natural to term this transfer of the output signals to the subsets, by analogy with supercontrol, the <u>subcontrol operation</u>.

However, the subcontrol operation defined in this way does not lead to a unique determination of the output signals, since neuron R from K appears in not one, but, generally speaking, several subsets of the various neurons from M(Q). For the elimination of the resulting ambiguity we note that among the subsets H of all the neurons from M(Q), to which the neuron is subordinate, the neuron P of interset to us (corresponding to the union of the properties I and J, as they were defined above) will be subordinate to all the remaining neurons of this subset. It is easy to see that the property $I \cup J$ will in this case have the highest unconditioned frequency among the properties corresponding to all the neurons from H. As a result of equation (108) this means that the output signal of the neuron P is the highest among the output signals of all the neurons from the subset H.

Thus, to ensure error-free functioning of the machine the subcontrol operation must be supplemented by still another rule: if as the result of the subcontrol operation several different output signals are transferred to some neuron, the largest of them must be selected.

Let us emphasize once again that the subcontrol operation is not applied to the neurons whose output signals are determined from the condition of direct srimulation or on the basis of the use of the supercontrol operation.

The unitary machine with all the described additions and alterations in the neuron functioning laws is termed a <u>conditional probability machine</u>. This name emphasizes the fact that the conditioned frequencies computed by the machine with sufficiently long training histories, which we will assume are usually performed using the independent trials scheme, tend with arbitrarily high confidence level to the corresponding conditional probabilities of some properties with respect to the others. The conditional probability machine is used for the simulation of the processes of the development and decay of the so-called conditioned reflexes. Let us assume that throughout the entire training history of the machine the property J appeared almost always together with the other property I. In that case the conditioned frequency p(J/I) of the property J with respect to the property I will be close to unity. If now property I appears without property J, then the reaction (output signal) of the neuron Q, corresponding to the property J, will differ little in its intensity from the reaction of the neuron P, which corresponds to the property I, and consequently is subject to the direct stimulation from the direction of this property. We will say in this case that in the machine there was developed a conditioned reflex for the property J with relation to the property I.

If after the development of the indicated reflex it is not reinforced over the course of a sufficiently large number of steps in the succeeding training history (i.e., the property I appears without the property J) then the conditioned frequency p(J/I) will diminish and can in the course of time become a negligibly small quantity. With the next appearance of the property I without the property J the reaction of the neuron Q will be very slight. In this case we shall speak of the decay of the corresponding reflex.

These processes of the development and decay of the conditioned reflexes, at least from the purely superficial view, are quite similar to the analogous processes taking place in the living organisms, in particular in the human nervous system. At the same time there are several differences. One of the essential differences is that in the scheme we have described the rate of development and decay of the conditioned reflexes in the very beginning of the training process is too high and at the end of the training process this rate is too low. This situation can be rectified by replacement of the unconditioned and conditioned frequencies by the so-called pseudofrequencies.

The unconditioned pseudofrequency of any given property I is the quantity r, included between zero and one, which increases with the appearance of images with the property I and decreases with appearance of images not having the property I. The quantity r must tend to one if, beginning with some moment, all the terms of the training sequence have the property I, and must tend to zero if all the terms of the training sequence, beginning with some moment, do not possess the property I (we assume here that the training sequence can be complemented with new terms in the course of an arbitrarily long period of time).

This definition is satisfied, in particular, by the unconditioned frequency, which can therefore be considered as one of the possible methods for the specification of the unconditioned pseudofrequency. However, it is simpler and more convenient to consider as the unconditioned pseudofrequency some property of the quantity $r_n = r_n(I)$, given by the recurrence relations

$$r_{n+1} = \begin{cases} r_n + \alpha(1 - r_n) \\ r_n - \beta r_n \end{cases} \quad (n = 0, 1, 2, \ldots), \qquad (109)$$

where the quantities α and β are positive constants which are strictly less than unity. If at the $(n + 1)$th step of the training there appears the property I, then we use the upper line of the indicated relations, otherwise we use the lower line. The initial value r_0 of the quantity r_n can be selected arbitrarily on the interval (0, 1).

If now in the conditional probability machine described above we replace the unconditioned frequency of the properties by the pseudofrequencies calculated with the aid of equation (109) and leave the neuron functioning method as before, then by selecting suitably the quantities α and β, we can approach much closer to the imitation of the biological processes of the development and decay of the conditioned reflexes than we can in the case of the original conditional probability machine. The conditioned frequencies of the properties will as before be computed using equation (107), however, in place of theunconditioned frequencies in the right side of this equation there will appear the unconditioned pseudofrequencies. Therefore equation (107) will now give not the conditioned frequency, but some quantity which it is natural to term the conditioned pseudofrequency of one property with relation to another.

We can, moreover, by defining in a somewhat different way the concept of the conditioned pseudofrequency improve the conditional probability machine so that it will immediately determine the conditioned the conditioned pseudofrequencies of the properties without preliminary calculation and memorizing of their unconditioned frequencies or pseudofrequencies. To do this we introduce into the unitary classification machine paired directed connections between the neurons. Each such connection is assigned some weight, which can take any real values on the interval (0, 1). These weights can vary at every step of the training. We shall denote the weight of the connection between the neurons P and Q in the nth training step by $\lambda_n(P, Q)$. We note that the weights $\lambda_n(P, Q)$ and $\lambda_n(Q, P)$ are not necessarily equal to one another.

The law of variation of the connection weight is specified by the following relations, defined for all values of n = 0, 1, 2,... :

$$\lambda_{n+1}(P,Q) = \lambda_n(P,Q),$$

if at the (n + 1)th training step the neuron P is not stimulated:

$$\lambda_{n+1}(P,Q) = \lambda_n(P,Q) + \alpha(1 - \lambda_n(P,Q)), \tag{110}$$

if at the (n + 1)th training step both neurons P and Q are stimulated;

$$\lambda_{n+1}(P,Q) = \lambda_n(P,Q) - \beta\lambda_n(P,Q),$$

if at the (n + 1)th training step neuron P is stimulated and neuron Q is not stimulated. (110)

Here, just as in equation (109), α and β are positive constants which are strictly smaller than unity and the initial weight $\lambda_0(P, Q)$ can be chosen arbitrarily on the interval (0, 1).

In the purely qualitative aspect the weight $\lambda_n(P, Q)$ behaves exactly like the conditioned frequency $p_n(J/I)$ of the property J, corresponding to the neuron Q, with respect to the propert I, corresponding to the neuron P. Actually, with simultaneous appearance of the properties I and J there is an increase of both the quantity $\lambda_n(P, Q)$ and of the quantity $p_n(J/I)$. Both these quantities diminish with the appearance of the property I without the simultaneous appearance of the property J. If the first situation (simultaneous appearance of the properties I and J) repeats itself several times in a row, then the quantities $\lambda_n(P, Q)$ and $p_n(J/I)$ tend to unity. With numerous repetitions of the second situation (the appearance of I without J) both these quantities tend toward zero. Therefore it is natural to term the quantity $r_n(J/I) = \lambda_n(P, Q)$ the conditioned pseudofrequency of the property J with respect to the property I.

If we say that in the conditional probability machine the neurons which are not directly stimulated must deliver as their output signals not the conditioned frequencies, but the conditioned pseudofrequencies of the properties corresponding to them with respect to the maximal positive property I of the image being considered at the given step, then for the accomplishement of this it is sufficient to transfer from the neuron P, corresponding to the property I, the output signals $\lambda_n(P, R)$ to all the neurons R which are not directly stimulated. Here it is convenient to consider that the neuron P sends along all the connections of the form (P, R) its unit output signal, which is attenuated along the corresponding connection as the result of multiplication by the quantity $\lambda_n(P, R)$, which by the condition is always less than unity.

It is not difficult to see that this mechanism for the formation of the conditioned reflexes has still another essential deficiency. This is that, as the result of the assumptions we have made, the conditioned reflexes are formed only with respect to the entire images and not to their individual properties. As is known, in the case of the biological systems the situation is different. Moreover, the reflexes are most frequently formed precisely with respect to the partial (not maximal) properties of the images.

We can, it is true, in the conditional probability machine of either of the types described above fix once and for all the property I with relation to which the conditioned frequencies and pseudofrequencies are computed. In this case the property may not be the maximal positive property, and the conditioned reflex will be developed precisely with respect to the property of the images and not to the image itself. All the definitions made above of the laws of the functioning of the conditional probability machine are aslo applicable to this case, only in this case there is no need to make a special search for the neuron corresponding to the maximal positive property of the image being considered: in the case of the appearance of the property I the role of this neuron will always be played by the neuron corresponding to the property I. However, in the case of nonappearance of the propetry I all the neurons (with the exception of the neurons which are directly stimulated) must, by definition, deliver not the conditioned, but the unconditioned frequencies (or pseudofrequencies) of the properties corresponding to them.

In his original definition Attlee considered precisely this method of functioning of the conditional probability machine. However, in avoiding one deficiency we come upon another, and again obtain a system which is significantly different from the biological systems, which are capable of developing conditioned reflexes with respect to several properties, and not just to one of them.

In order to avoid these last deficiencies, it is sufficient to remove the limitation in the last of the schemes which we have described which permits formation of the output signals of the neurons which are not directly stimulated only from the output signal of the single neuron P. In place of this we make the following assumption.

To every neuron Q which is not directly excited we transfer the signals $\lambda_n(P_i, Q)$ from all the directly stimulated neurons $P_i (i = 1, 2, \ldots)$. The output signal of the neuron Q will be the signal from the number of signals $\lambda_n(P_i, Q)$ $(i = 1, 2, \ldots)$ which has the largest magnitude.

The device which realizes this mechanism for the generation of the neuron output signals and for the alteration of the weights of the connections in accordance with equation (110) is termed a <u>conditioned reflex machine</u>. We can hope that it reflects many important properties of the real neural networks which constitute the neural systems of the animals and even the human neural system.

In the real neuron networks, of course, there are not all the connections required by the complete circuit of the conditional reflex machine. Further improvement of the laws of the variation of the weights of theconnections is also possible. In particular, the first of equations (110) should apparently be replaced by the equation $\lambda_{n+1}(P, Q) = = (1 - \gamma)\lambda_n(P, Q)$ where γ is a very small positive constant. After this refinement the law of the variation of the weights of the connections will reflect the property of the connections to diminish in the course of time even in the absence of cases of direct nonreinforcement of the reflex which is reflected by this connection.

If in the conditioned reflex machine we drop the requirement for completeness, i.e., the presence of neurons for all positive properties without exception, for all possible images without exception, then such incomplete conditioned reflex machines can be used for operation with a large number of receptors. On this basis we can possibly use the conditioned reflex machines for the solution of the problems of the recognition of visual patterns which we have been considering in the preceding sections. To obtain effective results in this direction it is necessary to make a purposeful selection of those properties to which the neurons in the indicated incomplete machine will correspond.

Still another problem associated with the classification systems is of interest. In the classification systems described so far all the images are perceived simultaneously with the images themselves. In many cases, however, we must deal with images whose properties are manifested gradually, in the course of the training. Moreover, in these cases the images, as a rule, are so remote from their visual prototypes that we shall term them concepts rather than images.

Such a situation arises in the design of self-improving systems for the recognition of the meaning of phrases (see Glushkov, Trishchenko and Stogniy [30]). In the simplest case, when we consider phrases consisting of only a subject and predicate, the concepts being recognized may be considered to be the nouns selected as the subjects, and their properties may be the possibility or impossibility of their meaningful combination with the various verbs appearing as predicates.

If the verb list is fixed and includes n different verbs, then each concept (noun) can be characterized by a line of compatability with these verbs, having the length _n_. At the _ith_ location on this line there will be one or zero in accordance with whether the combination of the considered noun with the _ith_ verb of the given list (i = 1, 2, ..., ..., n) is meaningful or not.

We shall call these lines the _verb lines_. The problem of the self-improving system in this case will include the prediction of the largest possible number of properties of the considered concepts on the basis of a limited experiment. If the experience (training history) consists in communicating to the mentioned system, one after another, meaningful combinations of randomly selected nouns and verbs from the given lists, then in the course of the arrival of such information certain places of the verb lines of the concepts (nouns) which we have selected will gradually be filled with ones. In this case the training problem can be treated as a problem of the reconstruction of the structure of the unitary machine for the classification of the properties of the selected concepts. For the solution of this problem it is natural to organize the process of the combination of concepts which are compatible with the same verbs into classes which correspond to new generalized concepts which may not be present in the original list. For example, as the result of combining several concepts (say, "father," "son," "student," and "professor" and so on) with respect to the criteria of compatability with the verbs "live" and "think" there arises the concept of "human." If after the formation of a particular class it is seen that certain of its representatives have some new elementary property (for example, the possibility of combining with the verb "go") then this property can be extended to the entire class (i.e., to all the concepts occurring in this class). Of course, errors may occur in this extension of the properties. To eliminate the resulting errors we introduce the process of independent composition by the machine of new phrases as the result of combination of other (randomly selected) concepts from the considered class with the verb whose compatibility was extended over the entire class. The sense (or nonsense) of these combinations must be communicated to the machine by the human teacher.

Thanks to the existence in language of connections similar to the connection which is described by the statement of the type "almost all those who think can also speak," the described process with judiciously chosen methods of formation of the classes, extrapolation of the properties, and composition of new phrases makes it possible for the machine to perform the correct separation of phrases into meaningful and meaningless with high probability. Here is is of essence that this separation be performed for all phrases which can be constructed from the given sets of nouns and verbs. In particular, they may include phrases which have not been constructed by the machine as questions for the teacher and those which were not communicated by the teacher to the machine in the training process.

Experiments on the training of a machine to recognize the meaning of phrases, not only of the simplest construction just described, but also phrases having a more complex structure, have been conducted successfully in the Institute of Cybernetics of the Academy of Sciences of the Ukrainian SSR [30]. With various assumptions relative to the structure of the language (in the present case — the sets of verb lines) we can make estimates for the learning rate in the realized algorithms similarly to the way this was done for the α-perceptrons in the preceding section. However, the corresponding arguments are quite cumbersome and considerably less graphic than in the case of the perceptrons.

§9. SELF-ORGANIZATION AND SELF-ADAPTATION. METHODS OF SOLUTION OF COMPLEX VARIATIONAL PROBLEMS

In §1 of the present chapter we introduced the concept of the algorithm system as the natural form in which the properties of self-organization and self-improvement are combined. Let us consider ． concept in more detail. For most of the problems encountered in practice it is advisable to differentiate self-organization itself and the so-called self-adaptation, which is the simplest case of self-improvement. More precisely, we shall differentiate the simplest type of self-improvement on the basis of self-adaptation and the higher type of self-improvement on the basis of self-organization.

The difference which is involved here is that self-improvement on the basis of self-adaptation assumes the variation of only certain numerical parameters in the operational algorithm, while self-organization is associated with the variation of the structure of the algorithm itself. Of course, this difference is to a considerable degree artificial since with suitable writing of the algorithms the variations in the algorithm structure can be reduced to variations of the numerical parameters. If, for example, a numeration of all the algorithms of the considered class is accomplished, then any alteration of the algorithm reduces to a change of the corresponding number, which can be considered as a numerical parameter. However, in spite of the relativity of the difference between the use of some fixed form of writing of the algorithms (algorithmic language) this difference can be drawn quite sharply.

We shall present some examples of self-adaptation and self-improvement of the structure of algorithm schemes. As the first example let us consider the case frequently encountered in practice of self-adaptation which carries the special name of extremal regulation. The essence of the problem of extremal regulation consists in the delivery to the regulation system of those values of certain parameters x_1, x_2, ..., x_n such that the specified function $f = f(x_1, x_2, \ldots, x_n)$ of these parameters takes on an extremal (maximal or minimal) value. Here the function $f(x_1, x_2, \ldots, x_n)$ also depends on certain other parameters y_1, y_2, ..., y_m which vary regardless of our desires and over which direct control is not possible. As the result of their change, the values of the parameters x_1, x_2, ..., x_n, which provide the desired extremum of the function $\underline{f}$ cannot be selected once and for all – they must be altered along with the change of the parameters y_1, y_2, ..., y_m.

In practice we most frequently encounter the case when finding the extrema by the conventional methods (with the aid of equating the partial derivatives of the function $\underline{f}$ to zero and the solution of the resulting system of equations) is impossible or inexpedient. The reason for this may be, for example, the absence of an analytic expression for the function $\underline{f}$. The question arises of what methods can be used for the solution of the problem of self-adaptation in this situation. One of the universal methods for the solution of this problem in this case is the so-called <u>method of steepest descent</u> (or steepest ascent).

The method of steepest descent (ascent) serves for finding the minimum (maximum) points of a function of many variables with the aid of the development of a special process for sequential approximation to these points. Let $f(x_1, x_2, \ldots, x_n)$ be any differential function of $\underline{n}$ variables. In the space of these variables we select the arbitrary point $M(a_1, a_2, \ldots, a_n)$ and find the approximate values of the partial derivatives $f_i = f'_{xi}(a_1, a_2, \ldots, a_n)$ at the point M from the equations

$$f_i \approx \frac{1}{\Delta}(f(a_1, a_2, \ldots, a_{i-1}, a_i + \Delta, a_{i+1}, \ldots, a_n) - $$
$$- f(a_1, a_2, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n)) \qquad (i = 1, 2, \ldots, n),$$

giving all the variables in turn the same increment Δ. Let us find out what increments must be given simultaneously to all the arguments in order to approach the extremum point to the maximal possible degree using only the values of the function f and its derivatives at the point M.

This latter condition is quite essential, since without it the question would be solved trivially; the increments of the variables could be such that they would lead us directly to the extremum point. However, we do not know the extremum point and we are required to resolve the question on the approach to it on the basis of the information and the behavior of the considered function in the neighborhood of the selected arbitrary point $M(a_1, a_2, \ldots, a_n)$. Denoting the desired increments of the variables $x_1, x_2, \ldots, x_n$ at the poing M by $\Delta_1, \Delta_2, \ldots, \Delta_n$ respectively and using the equation for the total differential, we obtain for the increment Δf of the function f at the point M the approximate equation

$$\Delta f \approx f_1\Delta_1 + f_2\Delta_2 + \ldots + f_n\Delta_n.$$

If we agree to take a step in the direction of the extremum point (in the $x_1, x_2, \ldots, x_n$ variable space) only of some one constant length r, then still another equation is added to the equation for the equation for the for the increment of the function f

$$\Delta_1^2 + \Delta_2^2 + \ldots + \Delta_n^2 = r^2. \qquad (111)$$

We must choose the quantities $\Delta_1, \Delta_2, \ldots, \Delta_n$ so that with satisfaction of equation (111) the function Δf will reach a maximal (with account for the sign) value. Using the method of undetermined Lagrange multipliers, the question reduces to finding the extremum of the function

$$F = \sum_{i=1}^{n} (f_i \Delta_i - \lambda \Delta_i^2) + \lambda r^2$$

of the variables $\Delta_1, \Delta_2, \ldots, \Delta_n$. Differentiating with respect to Δ_i and equating the resulting partial derivatives to zero, we obtain the system of equations

$$f_i - 2\lambda \Delta_i = 0 \quad (i = 1, 2, \ldots, n), \tag{112}$$

which must be supplemented, of course, by equation (111). From equation (112) we find that

$$\Delta_i = k f_i, \tag{113}$$

where k is the coefficient of proportionality, common for all i = 1, 2, ..., n.

Depending on the choice of the sign and the coefficient of proportionality k, equations (113) determine two opposite directions along which movement from the point M will lead to the (in the vicinity of point M) most rapid increase (with $k < 0$) of the function f. These directions are termed respectively the directions of steepest ascent and steepest descent at the considered point M. The magnitude of the advance r in either of the indicated directions is termed respectively the ascent or descent gradient step at the point M.

Depending on whether we are required to find the maximum or minimum point of the considered function f, we select one of these directions (sign of the parameter k in equations (113)) and perform the movement in the selected direction (determined by the choice of the magnitude of the parameter k) until the function f changes the nature of its growth in this direction, i.e., switches from increase to decrease, or, vice versa, from decrease to increase. In other words, the maximal advance is made in the selected direction which provides for variation of the function f in the desired direction (in the direction of decrease with search for the minimum point of the function f and in the increasing direction with search for the point of its maximum).

Denoting by the letter N the point obtained as the result of this movement, we perform the same operations with it that were described for the point M. As a result we obtain the new point P, and so on. If the function _f_ is sufficiently smooth, then as the result of the process described, continuing it sufficiently long, we arrive at an arbitrarily small neighborhood of some stationary point of the given function, i.e., that point at which all the partial derivatives of the function are equal to zero (of course, this will be true only in the case when the given function has stationary points) or at the neighborhood of a point of the boundary of the domain of definition of the function _f_ corresponding to some local extremal (in the given domain) value of the function _f_.

The desired point of the (absolute) extremum of the function _f_ with the assumptions made is among the indicated points. However, there is no guarantee that the application of the methodology described above will lead the first time to the point of absolute extremum of the given function. Therefore, we must by varying the initial point M find (by the method described above) new stationary points of the function, so that as a result of subsequent comparison of the values of the function at these points we can select from among them the desired extremum point.

In practice a different route is generally preferred: we select a random series of points M_1, M_2, ..., M_k in the domain of definition of the function _f_, and from them we vary that one at which the function has the maximal (in the case of the problem of finding maximum points) or minimal (in the case of finding minimum points) value. Starting from the point thus chosen, we perform the steepest descent or steepest ascent using the methodology described above. With sufficiently large _k_ (depending on the selection of the function _f_) with a probability arbitrarily close to unity there can thus be found the point of absolute (and not some local) extremum.

One of the possible variants of the search for the absolute maximum of the function of two variables is shown in Fig. 15. In this figure the function is specified by its contour lines (lines of equal level), M_1, M_2, M_3 denote the randomly selected initial points (the point M_2 is the highest of them), and N, P, Q denote the sequential series of points obtained from the point M_2 using the steepest ascent method. In this example, after only three steps of the steepest ascent we arrive at the point of absolute maximum of the considered function.

The algorithm system which resolves the problem of self-adaptation consists of the operational algorithm A which, receiving the input word (value of the function $f(x_1, x_2, \ldots, x_n)$) defining the criteria of the regulation quality and, possibly, the values of some other quantities, delivers an output word consisting of the coordinates of some point $M(a_1, a_2, \ldots, a_n)$ which coincides in the case of the stationary regime (invariance of the function $\underline{f}$) with the point of absolute extremum of this function. In the case of the nonstationary regime (variation of the function $\underline{f}$) there comes into play the algorithm B which performs the search for the point of the absolute extremum of the altered function $\underline{f}$ by some method (the method of steepest descent or ascent, for example) and replaces by the coordinates of this point the parameters a_1, a_2, ..., a_n, put out by the operational algorithm A.

The described system, consisting of the algorithms A and B, can be considered as a self-adaptive system of algorithms.

Let us now consider an example of self-improvement with variation of the structure of the operational algorithm. Let us assume that the operational algorithm must, from the various numerical values of the coefficients $\underline{p}$ and $\underline{q}$ of the reduced quadratic equation $x^2 + px + q = 0$

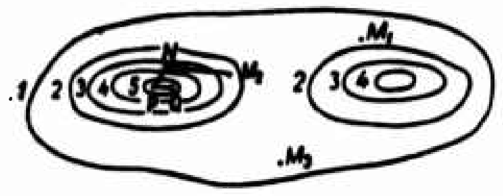

Fig. 15

deliver the roots of this equation, although in the beginning of the operation this algorithm is not yet known. Since the reduced quadratic equation is resolved using the known equation $x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$, the desired algorithm can be sought in the class of equations constructed with the aid of the operations of addition, subtraction, multiplication, division and extraction of the square root, using the letters $\underline{p}$ and $\underline{q}$ and whole numbers. All such equations can be numbered, after first arranging them in order of increasing complexity: with increase of the number of the equation there is an increase, generally speaking, of the number of operations used in the equation and an increase of the maximal magnitude of the integral parameters contained in it.

Initially, one of the simplest equations is selected as the operational algorithm, say the equation $p + q$. Taking successive values of the coefficients $\underline{p}$ and $\underline{q}$ ($p = 3$, $q = 2$, for example) the operational algorithm delivers the corresponding values of the root or roots (in the present case $x = p + q = 5$). The training algorithm makes a verification of thesolution obtained by substituting the value of the root found in theoriginal equation. If these values of the roots satisfy the equation, then the operational algorithm is retained unchanged. If however, the solution is found to be incorrect (as in the case just considered) then thenext equation in order is selected as the operational algorithm.

It is easy to see that with the described organization of the operational and training algorithms, after a finite number of unsuccessful attempts there will be established the correct equation for the solution of the quadratic equation. Since in the process of the search

there is a change, generally speaking, of the structure of the algo-
r ,form of the equation) and not simply of thenumerical parameters, then according to the terminology which we have adopted the constructed algorithm system is self-improving on the basis of self-organization, i.e., it is a higher type of self-improvement in comparison with self-adaptation.

In the example considered the search for the required working algorithm is performed by simple sorting of all the algorithms of the *a priori* given class. We can, of course, not make use of preliminary determination of some special class of algorithms, but rather perform the sequential sorting in the class of all algorithms written in a particular fixed algorithmic language (for example, in the language of the normal algorithms), however in this case the search time as a rule, is considerably longer.

Normally the systems for such a search are realized on high-speed electronic computers which perform several thousand operations per second. With this speed the solution of the problem described (finding the equation for the solution of the reduced quadratic equations) takes little time. However, with further complication of the sought operational algorithms the search time, based on the sorting of all the variants, increases so rapidly that the realization of such a search in a reasonable time, even using the high-speed computers, becomes impossible.

In this case we resort to multistage search: we first look for some sufficiently simple component parts (blocks) of the desired operational algorithm, and then use various combinations of the constructed blocks. The blocks themselves can be built up from still smaller blocks, so that the process of further division of the search into individual steps can be continued even further. The multistep search systems permit the construction of very complex self-organizing systems which are analogous to the creative search systems used by man.

Without going into further detail concerning self-improvement on the basis of self-organization, we shall concentrate our attention on the problems associated with self-adaptation.

The method of steepest descent (or ascent) described above is also a certain sort of search, In this search we use a definite <u>strategy</u> (or <u>tactic</u>) for the reduction of the sorting of the various variants leading to the problem solution. In the case considered the search strategy reduced to the use of the information on the <u>local</u> properties of thecorresponding function <u>f</u>, which we term the <u>estimating</u> function (the values of this function can be considered as an estimate of the quality of theapproximation to the required solution which is found at a particular step of the search).

If the number of parameters (arguments of the estimating function) is very large, various difficulties arise in the use of the method of steepest descent (ascent) in the simplest form described above (cycling on secondary minima or maxima, excessively slow rate of advance toward the absolute extremum, etc.). In order to eliminate these difficulties we introduce various alterations and improvements in the local search strategy described above.

The simplest changes are associated with the selection of the particular descent (ascent) gradient step. In particular, it is desirable to perform the advance in a given direction, not until the increment Δf of the estimating function changes sign, but only until the relative magnitude of this increment $\Delta f/f$ is less (in modulus) than some (in modulus) then some <u>a prior</u> fixed quantity termed the gradient test.

In many cases we can divide the extimating function <u>f</u> into two classes, so that variations of the variables of the first class lead to relatively large variation of the value of the function f, while the variations of the variables in the second class alter this value to a considerably lesser degree. The methods described above provide too low a rate of advance toward the extremum in the directions corresponding to the variations of the variables of the second class. Figuratively speaking, we can perform a rapid descent (in the direction corresponding to the variations of the variables of the first class) to the bottom of some "gully" and then wander more or less randomly about its bottom without getting any closer in practice to the extremum point.

To eliminate this deficiency Gel'fand and Tsetlin [18] proposed a special method which they termed the gully method. The essence of the method is that on the "slopes of the gully" there are selected two points (X_0 and X_1) rather than one. From these points there is performed a steepest descent to the "bottom of the gully" as a result of which there arise two new points A_0 and A_1. Connecting the points A_0 and A_1 with a straight line, they perform in the direction of thid line (in the direction of reduction of reduction of the estimating function) the so-called gully step whose magnitude is usually considerably larger than the magnitude of the descent gradient step. This step leads to a new point X_2, from which we again perform a steepest descent to the point A_2, located on the "bottom" of the gully. In the direction defined by the points A_1 and A_2 we again make a gully step leading to the new point X_3. From it we again perform a steepest descent to the "bottom of the gully" and so on.

We have described the method for the descent (i.e., for finding the minimum of the estimating function f). Of course the corresponding constructions are applicable to the ascent (finding the maximum of the function f). In this case in place of the descent into the "gully" we perform an ascent to the "ridge" and the further advance, not along the "bottom of the gully" but along the "crest of the ridge."

All the described descent (ascent) methods relate to the class of methods for finding extrema of functions or functionals which we combine under the name of variational methods. In the majority of the cases of interest for cybernetics, particular limitations of the possible values of the arguments of the estimating function f take on considerable importance. In this case the sought extrema may be reached on the boundaries of the domain of definition of the function f rather than within the domain. The descent (ascent) methods considered above are in principle also suitable for finding such "boundary" extrema. However, in several particular cases certain special variational methods are far more effective.

Among this sort of methods are the so-called linear programming (or linear planning) methods. These methods are used in the case when the estimating function f is a linear function (polynomial of first degree): $f = a_1x_1 + a_2x_2 + \dots + a_nx_n + a_0$ and the boundaries of the domain of definition of the variables are composed of hyperplanes, i.e., surfaces specified by equations of the first degree. In this case the domain of definition is a multidimensional polyhedron (not necessarily finite), all points of which polyhedron satisfy the system of lenear inequalities of the form $b_{i_1}x_1 + b_{i_2}x_2 + \dots + b_{i_n}x_n + b_{i_0} \geq$ ≥ 0 $(i = 1, 2, \dots, m)$. The signs of the inequalities can, of course, reverse with a change of the signs of the coefficients b_{ij}.

It is not difficult to see that the estimating function f takes extremal value in one or several vertices the extremal value is taken to be the function f on the face (generally speaking, multidimensional) passing through all these vertices. Therefore we can find the desired extremal value by sorting one-by-one all the vertices of the polyhedron, however with a large number of variables this method is extremely cumbersome and not suitable in practice. Far more effective methods for the solution of linear programming problem have already been developed. These methods provide for the use not of the linear inequalities, but rather the linear equations to which any inequalities can be reduced by the introduction of new unknowns. With this introduction the inequalities $\sum_{j=1}^{n} b_{ij}x_j + b_{i_0} > 0$ are replaced by the equation $\sum_{j=1}^{n} b_{ij}x_j + b_{i_0} = y_i$, where y_i are the new unknowns which can take only the nonnegative values ($i = 1, 2, \ldots, m$).

In practice we most frequently encounter the following statement of the linear programming problem, which we shall term the canonical form.

Given the system of m linear algebraic equations with the unknowns $\sum_{j=1}^{n} a_{ij}x_j = b_i$ $(i = 1, 2, \ldots, m)$. Required to find that nonegative (all $x_j \geq 0$) solution of this system for which some fixed linear form (estimating function) $f = \sum_{k=1}^{n} c_k x_k$ takes the smallest possible value.

We shall describe one of thepossible effective methods for the solution of this problem which is usually termed the simplex method. In the use of the simplex methods we perform sequential transformations of the given system of equations $\sum_{j=1}^{n} a_{ij}x_j = b_i (i = 1, 2, \ldots, m)$ until it is reduced to some special form. The system is first transformed all the free terms b_i are made nonnegative (if $b_i < 0$, it is sufficient to change the signs of both sides of the ith equation); then the equations are rewritten in the form $0 = b_i - \sum_{j=1}^{n} a_{ij}x_j$ $(i = 1, 2, \ldots, m)$. If in the resulting system there is the variable x_k appearing only in one equation, say the pth, and having the positive coefficient a_{ik}, then that variable takes the name basic, and the corresponding equation is solved relative to this variable. Identifying all the basic variables, designated by the letters $x_1, x_2, \ldots, x_{k_0}$, we reduce our system to the form

$$\begin{aligned} x_k &= b_k - \sum_{l=k_0+1}^{n} a_{kl}x_l \quad (k = 1, 2, \ldots, k_0); \\ 0 &= b_i - \sum_{l=k_0+1}^{n} a_{il}x_l \quad (i = k_0+1, k_0+2, \ldots, m). \end{aligned} \tag{114}$$

The equations of the second group (not solved relative to x_k) are termed the 0-equations (it is not impossible that all the equations of the systems will be in this group). The purpose of the further transformations consists in finding some nonegative solution of the system (114). These transformations reduce to the sequential (multiple, generally speaking) repetition of the cycle consisting of the following steps (see [6]):

1. We find the 0-equation for which the free term is strictly greater than zero (if there is no such 0-equation, then the problem is solved, since, setting $x_k = b_k$ ($k = 1, 2, \ldots, k_0$) and $x_i = 0$ ($i = k_0 + 1, k_0 + 2, \ldots, n$) we obviously find the nonnegative solution of the system (5)). Let this be the ith equation.

2. In the found (ith) equation we identify some positive coefficient a_{ij_1} (if all the coefficients a_{ij} in the ith equation are nonpositive, then the system (114) obviously cannot have positive solutions and, consequently, the posed linear programming problem is unsolvable).

3. In the same column with the identified coefficient a_{ij_1} (i.e., in the j_1th column) we find the so-called resolving coefficient $a_{i_1j_1}$, which is characterized by the fact that of all the relations b_i/a_{ij_1} with positive a_{ij_1} ($i = 1, 2, \ldots, n$) the ratio $b_{i_1}/a_{i_1j_1}$ has the minimal possible value.

4. The equation in which the resolving coefficient appears (i.e.,

the i_1th equation) is solved relative to the variable x_{j_1}, which after this is related to the class of basic variables, and the found expression for x_{j_1} is substituted into all the remaining equations (if the i_1th equation does not belong to the number of 0-equations, the variable x_{i_1} standing in its left side is excepted from the number of basic variables).

5. After the solution of the i_1th equation (relative to x_{i_1}) we again find the 0-equation with a positive free term and the entire operation described above is performed with it.

The described process of sequential solution is continued until all the 0-equations disappear. The desired nonnegative solution is obtained as the result of equating all the basic variables x_i to the corresponding free terms $b_i (i = 1, 2, \ldots, n)$ and all the nonbasic variables to zero. There are cases when the described process cycles and continues infinitely long without leading to any solution. In these cases we resort to variation of the selection of the 0-equation and the resolving coefficient in the 2nd and 3rd steps of the sequential solution process, which usually helps prevent cycling.

After termination of the sequential solution process, the found expressions for the basic variables are substituted into the estimating function $f = \sum_{k=1}^{n} c_k x_k$, as the result of which it takes the form $f = d - \sum_{j=r+1}^{n} d_j x'_j$. In the latter expression the summation extends only to the nonbasic variables, which (after corresponding numeration) are assigned numbers from r + 1 to n.

Then the solution process described above is applied to the system of equations for the basic variables obtained as the result of the first application of this process, which is written as

$$x'_i = b'_i - \sum_{l=r+1}^{n} a'_{il} x'_l \qquad (i = 1, 2, \ldots, r).$$

and to the new O-equation $0 = d - \sum_{j=r+1}^{n} d_j x'_j$. As the resolving coefficients we select only the coefficients a_{1j}. The process terminates after all the coefficients d_j in the O-equation become negative. Setting after this all the nonbasic variables equal to zero, and all the basic variables equal to the corresponding free terms, we obtain the required solution of the original linear programming problem. We note that in both the first and second applications of the sequential solutuion process all the free terms (with the exception of theterm d) remain nonnegative all the time.

Linear programming is widely used in problems for the optimal planning of transport shipments. Such applications of linear programming were first developed by Kantorovich [38]. Detailed substantiations of thesimplex method which we have described can be found in special nomographs devoted to linear programming.

We shall describe still another general scheme of the variational problems to which many problems of so-called dynamic programming (or dynamic planning) are reduced [7]. The essence of this scheme reduces to the following: in some (generally speaking, multidimensional) Euclidean space with the aid of certain limitations we identify a certain class of curves which we shall term trajectories. On the set of all possible trafectories there is given some estimating function (or, as we usually say, functional). The problem consists in finding the trajectory on which the value of this functional is greatest or least.

Let us consider one of the quite general numerical (approximate) methods of solution of the indicated problem developed by Mikhalevich and Shor the method of sequential analysis of variants. We shall discuss this method with application to one of the simplest cases when the basic space is two-dimensional, the class K of trajectories consists of all the piecewise-smooth curves connecting the two fixed points A and B of the space and contained wholly in some fixed finite region Q, and the estimating functional F has the property of additivity. The additivity property consists in the functional F being considered defined not only on the integral trajectories but also on any of their pieces (open subsets and their closures) and with combination of two disjoint pieces into one, the corresponding values of the estimating functional are added

$$F(L_1 \cup L_2) = F(L_1) + F(L_2).$$

The formulated conditions correspond to the dynamic programming problem in the Bellman formulation.

The first step in the method of sequential analysis of variants is the limitation of the class K: of all the trajectories connecting the points A and B we identify only certain broken lines. This is done by means of passing several sections (straight lines in the present case) perpendicular to the segment AB and intersecting it. The vertices of the broken lines considered can be located only on the selected sections. Further, each section (in the limits of the region Q) is approximated by a finite set of points (possible vertices of the broken lines). The density of the positioning of the points of the approximating set, and the density of the sections, is defined on the basis of the required accuracy of the problem solution. A graphic impression of therequired constructions is given by Fig. 16.

In Fig. 16 the boundary of the region Q is shown by the dashed curve, the sections are denoted by the Roman numerals, and the points of the sets which approximate the sections are denoted by Arabic numerals. If the number of points in the ith section is denoted by n_i and the total number of sections by m, then the total number of broken lines (N) corresponding to the postulated conditions will be determined, as is easily seen, by the product of the numbers $n_i : N = n_1 n_2 \ldots n_m$.

This quantity increases very rapidly with increase of the number of points and sections, the result being that the sorting of all the variants is practically impossible.

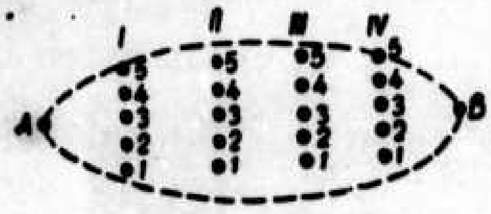

Fig. 16

However, we can abbreviate the sorting by using the following technique. Let us connect point A by segments of straight lines with all the points of the first section and calculate the value of the estimating functional F for all these segments. To each point j of the first section we assign the value F^1_j of the functional F on the segment connecting this point with the point A. For each point k of the second section we find the point j_k of the first section such that the value of theestimating functional F on the broken line $Aj_k k$ will be smallest in comparison with its values on all the other permissible broken lines connecting the point A with the point k. Since the functional F is additive, the question reduces to the minimization of the sum $F_{j_k} + F(j_k, k)$, where $F(j_k, k)$ is the value of the functional F on thesegment $[j_k, k]$. The corresponding (minimal among all possible values) value of the functional F on the broken line $Aj_k k$ is denoted by F^2_k. To find it we make use of the already found values of the functional F^1_j at the points of the preceding (first) section, which of necessity will be minimal here, and the sorting is limited to only all possible segments connecting the points of the first and second sections.

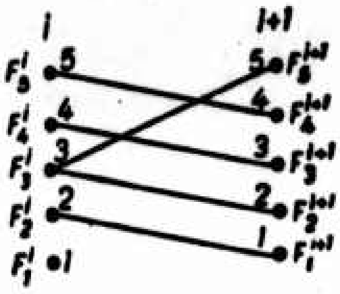

Fig. 17

Let us assume that for all the points p of the ith section there have already been found the minimal values F^i_p of the functional F on all the permissible broken lines connecting these points with the point A. Let us consider the portion between the ith and the (i + + 1)th sections (Fig. 17). For each point q of the (i + 1)th section we find the point p_q of the ith section such that the sum $F^i_{p_q} + F(p_q, q)$ is minimal. It is evident that the found minimal value of the indicated sum will be the minimal possible value of the estimating functional F on all the permissible broken lines connecting the point A with the point q. Recording this value F^{i+1}_q and forcing thepoint q to run through all the points of the (i + 1)th section, we find it possible to come to the consideration (on thebasis of completely analogous constructions) of the portion between the (i + 1)th and the (i + 2)th sections. However, for the consideration of the portion between the ith and the (i + 1)th sections we need to remember only the function $\varphi^i(q)$, assigning to each point q of the (i + 1)th section the point $p_q = \varphi^i(q)$ of the ith section with which it connects most favorably. In the case shown in Fig. 17,

$$\varphi'(1) = 2; \quad \varphi'(2) = 3; \quad \varphi'(3) = 4; \quad \varphi'(4) = 5; \quad \varphi'(5) = 3.$$

As as result of repetition of the indicated process we come, finally, to the consideration of the portion between the last section (mth) and the final point B. Finding for the point B the point $r = \varphi^m(B)$ of the mth section with which it connects most favorably, we can from the functions $\varphi^i(x)$ $(i = 1, 2, \ldots, m)$ which we recorded find the best trajectory (admissible broken line) $Ak_1k_2 \ldots k_i\ k_{i+1} \ldots k_mB$ connecting points A and B: the points of this broken line are determined sequentially (from right to left) with the aid of the relations $k_i = \varphi^i(k_{i+1})$ $(i = m, m - 1, \ldots, 1)$ where as the point k_{m+1} we select the point B. It is easy to see that the found broken line actually minimizes the estimating functional F. Here the sorting used in finding the broken line is obviously limited to only $n_1n_2 + n_2n_3 + \ldots + n_{m-1}n_m + n_m$

variants in place of $n_1 n_2 \ldots n_m$ variants with complete sorting (n_i is the number of points in the ith section).

The described method is generalized directly to the case of multi-dimensional spaces. The requirement for the additivity of the estimating functional F is also not strictly necessary. It is obviously sufficient to assume that for any initial piece APQ of the trajectory the value F(APQ) of thefunctional can be represented in the form F(APQ) = = f(F(A, P), F(P, Q)), where f(x, y) is a real function which does not decrease with respect to x for any value of y.

We note further that in the majority of the cases the sorting of the different variants of connection of the points of two neighboring sections can be considerably reduced as the result of various sorts of limitations (for example, the limitations on the maximal slope of the segments of the broken line with relation to the x axis in the two-dimensional case). In several cases the sorting can be reduced by the use of the property of the continuity of the estimating functional. More complex constructions in the method of sequential analysis of variants are associated with special systematization of the limitations and functionals which permits construction of algorithms for the search for the optimal trajectory by sequential detection and discarding of "unpromising" segments of the trajectories.

Chapter 5

ELECTRONIC DIGITAL MACHINES AND PROGRAMMING

§1. THE UNIVERSAL PROGRAM AUTOMATON

One of the most significant technical achievements of our time has been the creation of the universal program automata, i.e., the automatic information processors which make it possible to realize any algorithms. The modern universal electronic (digital) computers are such automata. It is interesting to note, as indicated by the name itself, that these machines were created for the purpose of automating computations, more exactely – for the automating of the performance of any computational algorithms. The term "universal" with application to these machine was understood by the creators of the first universal computers (and is still understood by many today) in the sense of universality with relation specifically to the computational algorithms.

However, since any algorithm can be reduced, as we noted in Chapter 1, to thecalculation of some partially recursive (arithmetic) function, the universality with respect to the computational algorithms turns out to be universality in general. This circumstance is of great practical and theoretical importance, since actually the basis of any field of human activity is the processing of information in accordance with particular, frequently very complex sets of algorithms.

The availability to us of the universal automatic information processors such as the modern universal electronic computers makes it possible, at least in principle, to automate any field of human activity which is based on the processing of information. This may be the solution of complex problems of a design nature, planning, production control, translation from one language to another, composition of music, playing chess, many others. It is curious that the tremendous possibilities inherent in the universal electronic machines not only were not recognized by their first designers, but were even disputed by some of them.

In this connection we must note still another error which is common among individuals who are not familiar with the theory of algorithms. The idea is prevalent that the amazing properties of the modern electronic digital machines are based on some specific characteristics of the elements used in these machines – the electron tubes, transistors, etc. In actuality, electronics in itself has no relation with their theoretical (qualitative) capabilities.

These essence of the matter lies in the specific control principle and in the set of operations which these machines can perform, while the elements from which they are constructed can be of quite varied physical nature and can, in particular, be purely mechanical. The electronic elements are used for the purpose of significantly increasing the operating speed of the computers, and also to improve their reliability (on the basis of some fixed number of operations performed by the machine). We note that the first universal digital computer (Mark-1) was built using electromechanical elements (electromagnetic relays) rather than electronic elements.

The control principle, which provides for algorithmic universality (capability of realizing any algorithm) of the modern universal digital machines is a development and generalization of the principle which is the basis of the algorithmic scheme of Post described in Chapter 1. Just as in the Post scheme, the information in the universal digital machine is stored in a memory which is divided into individual cells (memory cells). However, in contrast with the Post scheme, in each cell there may be stored not a single binary digit (0 or 1), but an entire word, composed of a considerable (usually 30-40) number of binary digits. We can, if convenient, consider these words as letters in some finite alphabet, however this alphabet will contain, as a rule, a very large number of letters $2^{30} - 2^{40}$.

Therefore we usually prefer to consider the contents of each memory cell not as an individual letter, but as a word in a binary alphabet. The binary digits (0 and 1) composing this word are usually termed (binary) places, and the word itself is termed a binary code, sometimes simply a (binary) number. We can, of course, consider the letters to be not the contents of the individual binary places, but some combination of thes places. For example, any binary code of length equal to three can be considered as a number in the octal notation system, designating by traids of binary digits the octal digits: 000-0, 001-1, 010-2, 011-3, 100-4, 101-5, 110-6, 111-7. Using not all, but only some four of the binary digits for the designation of the decimal digits, we can represent the binary codes consiting of such tetrads by numbers in the decimal notation system.

In addition to be replacement of the binary digits by the multiplace binary codes, there is a second essential difference in the organization of the memory (or the storage device) of the universal digital machine and the memory for the algorithms in the Post scheme. Being an abstract algorithmic scheme, the Post scheme assumed the existence of an infinite number of cells, or the existence of a memory of unlimited volume. At the same time, in the real technical devices, which the universal digital machines are the size of the memory is of necessity limited.

In the modern large universal digital machines the size of the high-speed memory does not exceed 100,000 cells (usually no more than 4-8000). This situation must be kept in mind, since it is closely related with the concept of the machine universality. Strictly speaking, for the possibility of the realization of any algorithm the universal digital machine must accomodate the writing (representation) of this algorithm in its memory. Since the representation of the algorithms can be arbitrarily long, for the actual capability of realization of any algorithms the machine memory must be infinite.

Keeping in mind, however, that an infinite memory cannot be realized in any technical device, it is customary to term a machine universal if the organization of its control and the set of operations are such that they would provide the possibility of the realization of any algorithm with the condition of unlimited size of the memory.

In practice the universality of the modern machines is provided by the fact that in addition to the high-speed (the so-called <u>operational</u>) memory device, it is also equipped with a relatively slow (the so-called <u>external</u>) memory devices which are capable of exchanging information with the operational memory device. The capacity of the external memory (usually composed of magnetic tapes) can be considered practically unlimited, which then determines (with the possibility of exchanging codes with the operational memory) the practical possibilith of the performance of any algorithm on the machine.

The sequence of operations performed by the universal digital machine is determined by the <u>program</u> established in its memory, which is an ordered finite set of <u>instructions</u> which can be considered as a natural generalization of the orders used in the construction of the Post algorithmic scheme. In contrast with the Post scheme in which the active cell is displaced with the performance of each succeeding order by no more than one step to the right or left, in the universal digital machine provision is made for the possibility of arbitrary variations of the position of the active cell from order to order. To do this, in each order there is introduced the number of one or several memory cells which are active with the performance of the given order.

The number of the memory cells in the universal digital machines are customarily termed addresses. The number of addresses in the orders of the modern universal digital machines (the number of memory cells which are active in the performance of any these orders) usually varies in the range from 1 to 4. Corresponding to this we differentiate single-address, dual-address, triple-address and quadruple-address orders.

Let us first consider machines with a quadruple-address system of orders, i.e. , those machines in which the maximal address level of the orders equals four. Different types of orders correspond to different operations which can be performed by the machine. The orders are usually recorded in the machine in the form of binary codes which can be stored in the machine memory (both operational and external).

We will assume that in each memory cell there can be contained either one order, also termed command or command word, or one information word. Just as was done above in the case of the information words, each command word (order code) can if desired be considered as a word in any finite (not necessarily binary) alphabet.

Any command word is divided into operational and address parts. In the operational part there is the code of the operation which is performed during the time of action of the order which is represented by this command word. All the orders of the same type have identical operational parts. In the address part of the order there are located the addresses of the cells which are active at the time of action of the order.

The operations performed by the universal digital machines are usually divided into several classes. The first class includes the arithmetic operations – addition, subtraction, multiplication and division. The four-address orders for the performance of the arithmetic operations usually have the following structure: in the operational part of the order there stands a code number designating the particular operation (for example, one – addition, two – subtraction, three – multiplication, four – division). The first two addresses in the order are used to indicate the addresses of the memory cells which store the numbers with which the operation is to be performed, i.e., the addresses of the addends in the case of addition, the addresses of the minuend and subtrahend in the case of subtraction, etc. The third address of the order shows the transfer of the result of the performance of the operation (sum, difference, product or dividend). Finally, the fourth address of the order is used to indicate the memory cell which stores the order to be performed following the given order.

The orders for the performance of the logical operations are constructed just as in the case of the arithmetic operations. The logical operations are as a rule two-place operations, performed place-by-place, i.e., separately for each pair of corresponding binary places which participate in the code operation. These include, for example, placewise conjuction (logical multiplication) and placewise disjunction (logical addition). There are also single-place logical operations on the codes. Such operations incluse right and left (logical) shifts which transform the binary code $x_1, x_2, \dots x_n$ into the codes $0x_1x_2 \dots x_{n-1}$ and $x_2x_3 \dots x_n0$ respectively.

A special role is played by the so-called control transfer operations, which serve for the variation of the order of performance of the program orders as a functions of the results obtained in the realization of the program. A typical control transfer operation (also termed the conditional transfer operation) is the so-called operation of conditional transfer on exact coincidence of words. The first two addresses of the order which realizes this operation are used for the indication of the memory cells from which the two words being compared with one another are taken. In the case of coincidence (quality) of these words the next order is taken from the memory cell indicated by the third address, and in the case of noncoincidence it is taken from the fourth conditional transfer order address. Conditional transfers of other forms are also possible, for example, conditional transfer on the basis of the sign of the difference of two words or on the basis of the sign of some one word (in the latter case, of course, it is sufficient to have three rather than four addresses in the conditional transfer order).

The memory of the universal digital machines is usually arranged to that with the selection (reading) of a word from any cell for the performance of a particular operation there takes place a sort of bifurcation of this word. One of its exemplars goes to the corresponding device for the performance of the operation, while the other remains in the cell from which the selection was made. With the writing of a new word into a particular memory cell the information previously contained in this cell is automatically destroyed (erased).

Taking account of the indicated properties of the memory of the universal digital machines, it is not difficult to note that for the performance of any Post algorithm it is sufficient to make use of only the operation of (algebraic) addition, one of the conditionald transfer operations, for example the operation of conditional transfer on exact coincidence of words, and the operation of machine stop.

Actually, we shall agree to operate with only some two information words p_0 and p_1, the first being identified with zero and the second with unity of the binary alphabet of any given Post algorithm A. Let us divide the memory of the considered universal digital machine into three parts. The first part consists in all of five cells: a_{-1}, a_0, a_1, b_0, b_1 in which there are placed the words – 1, 0, 1, p_0, p_1; in the cells of the second part there will be placed the program which simulates the program (scheme) of thealgorithm A; finally, the third portion of the machine memory M simulates the information tape of the algorithm A.

With operation of the algorithm A on a specific input word $\underline{p}$ on which this algorithm is defined, the original, intermediate and final information occupies only some limited (finite) part of the information tape, since the algorithm operates only a finite number of steps and at each step writes information in no more than one new cell. Therefore, if the machine memory M is sufficiently large we can place in its third part identified above the required portion of the information tape. The difficulties with the possible insufficiency of the memory size, in view of the assumption made above, should not be taken into account in the resolution of the question on the theoretical representability on the machine of particular algorithms.

Let us turn to the direct simulation of the orders of the Post algorithm A by the orders of the machine M.

As noted in §4 of Chapter 1, in the Post algorithms six different types of orders may be encountered. The order of the sixth type (stop) is simulated directly by the corresponding order of the machine M. The orders of the first two types (writing zero and unity in the cell being considered) are simulated by the orders of the machine M which accomplish the transfer of information from the cells b_0 or b_1

into the active cell indicated, for example, by the third address of the machine order. It is clear that this transfer can be accomplished by an addition order, in the first two addresses of which there is the pair of addresses of the cells a_0 and b_0 or of the cells a_0 and b_1, while in the third address theree is the address of the active cell (the fourth address, just as in the Post algorithm, is used for the indication of the address of the order which must be performed following the present order).

The Post order of fifth type is simulated, as it is not difficult to see, by the machine order for conditional transfer on the basis of exact word coincidence. It is sufficient to compare the word in the cell b_1 with the word in the active cell and transfer to one of the two orders on the basis of the results of this comparison.

Finally, each Post order q' of third or fourth type is simulated with the aid of a group of machine orders whose number is equal to the number of orders of first, second and fifth types in the considered Post algorithm A. Actually, let r' be any order of first, second or fifth type of the algorithm A. In view of what was said above, it is simulated by a single machine order, which we denote by $\underline{r}$. Let the Post order q' displace the active cell one unit to the right. In the case of the machine program this displacement can be accomplished only by means of variation by plus 1 of the addresses of all the active cells.

There are such cells, however, only in the machine orders which simulate the Post orders of first, second and fifth types. As a result of suitable numeration of the addresses we can, without losing generality, assume that the addresses of the active Post cells are written in some definite, let us say the last, address of the order. Considering the codes to be whole binary numbers, we come to the conclusion that for the alteration of the address of the active cell in the order r by +1 it is sufficient to add the code of the command r with the code plus 1, located in the cell a_1. The shift of the active cell to the left is accomplished analogously.

From what we have said it is clear that algorithmic universality of any program controlled digital automaton will be provided if with the aid of the operations performed by it we can accomplish the four operations:

1) the operation of the transfer of the contents of any memory cell to any other memory cell;

2) the operation of the addition of the code of an order located in any memory cell with constants which alter the value of the given (first, second, third or fourth) address of the order by plus 1 or minus 1;

3) the operation of conditional transfer on exact word coincidence;

4) the operation of (unconditional) machine stop.

In the case considered above, operations 1) and 2) are provided by the same machine operation – the operation of algebraic addition. Usually, however, in the universal digital machines these operations are separated, the second being termed the readdressing operation or command addition.

Of course, in addition to the indicated operations, in the set of operations of every universal digital machine there must be the operations of the entry and exit of the information from the machine, and aldo (in the case of the use of an external memory) the operations which provide for two-way exchange of information between the operational and external memory devices.

Let us consider the question on the ways of reducing the number of addresses of the orders. First of all it is not difficult to see that the fourth address, used for the indication of the succeeding order, can be eliminated by positioning the orders in the machine memory so that the address of the order to be performed following any given order p is always larger by unity than the address of the order p itself. In other words, the use of the fourth address becomes unnecessary under the condition that the order of arrangement of the instructions in the memory cells corresponds to the order of their performance by the machine.

Violation of the usual (natural) order of succession of instructions can occur only in the case when the instruction being performed is the conditional transfer command. As we noted above, in the four-adress system of instructions one address is used for the indication of the following instruction with nonsatisfaction of the condition on which the conditional transfer is based, and a second address is used for the indication of the following instruction with satisfaction of this condition. With replacement of the four-address system of instructions by a three-address system it is usually assumed that in the first case (nonfulfillment of the condition) after the instruction for conditional transfer there is performed the instruction written in the next memory cell in order, and only in the second case with fulfillment of the condition is one of the addresses used for the indication of the address of the instruction which must be performed next.

From this it follows that the fourth address can be made redundant not only in the ordinary instructions which do not alter the subsequent order of performance of the instructions, but also in the instructions for the conditional transfer. The resulting three-address instruction system is usually termed a system with natural succession of instructions, in contrast with the previously described four-address system with forced succession of instructions. The advantage of the latter system lies in the greater freedom which it offers in the question of the arrangement of the sequence of commands in the memory device which are used to control the operation of the machine. The advantage of the three-address system is the simplification of the structure of the instruction.

Further reduction of the number of addresses in the instructions can be achieved as a result of fixing some supplementary memory cell, usually structurally separated from the other memory cells of the machine. After the fixing of this cell a simple way is opened to the reduction of the number of addresses in the instruction to the natural minimum, i.e., to a single address. Let us clarify this method using as an example the addition operation, which requires the use of three addresses: the address of the addend a_1, the address of the augend a_2 and the address a_3 to which the sum is to be sent. With the aid of the fixing of the supplementary cell b_0 this three-address operation can be performed by means of the sequential performance of three single-adress operations – the operation of the transfer of the number from the cell a_1 into the (fixed) cell b_0, the operation of addition of the number contained in cell a_2 with the number in cell b_0 with subsequent writing of the result in cell b_0 and, finally, the operation of transfer of the number from cell b_0 into cell a_3. Since in this case only the addresses a_1, a_2, a_3 can change while the address b_0 is fixed once and for all, all three indicated operations are actually realized with the aid of single-address instructions.

The described method leads to the single-address system of instructions which is used in many of the modern universal electronic digital machines. Having the single-address system, it is not difficult to construct also the two-address system of instructions. In this case the second address can be used either for the indication of the address of the following instruction (two-address system with forced succession of instructions) or for the indication of the addresses of the number codes (information words) with which the operations are performed (two-address system with natural succession of commands).

Every device having a discrete memory which is divided into individual cells and whose operation can be controlled with the aid of a sequence of command words – instructions – which are arranged in certain of these cells is termed a program automaton and the indicated sequence of instructions itself is termed the program for the automaton operation.

If the set of operations (types of instructions) performed by the program automaton makes it possible to compose from them the operation of the transfer of the information words from any memory cell into any other memory cell, the operation of readdressing (alterations of the addresses in the instructions) by ±1, the operation of conditional transfer and machine stop, and if as the program of the automation there can be specified any finite sequence of operations from this set, then such an automaton is termed a universal program automation.

With an accuracy to the limitations introduced by the fixed memory size, the universal program automaton is capable of reproducing any algorithm with the condition of suitable coding of its input and output alphabets. This conclusion relates not only to the conventional algorithms, but also the random and self-altering algorithms (see §5 of the present chapter).

§2. STURCTURE OF THE MODERN UNIVERSAL PROGRAM AUTOMATA

The modern universal program automata consist of five different basic devices – the memory unit (MU), the arithmetic unit (AU), the control unit (CU), the input unit and the output unit (output). As we noted in the preceding section, the memory device (memory) serves for the memorizing and storing of the program for the automaton operation, and also of the initial, final and intermediate information. The input device serves for the input of the program and the initial information (conditions of the problem) into the automaton memory, the output device serves for the output from the memory of the final information (response to the problem posed to the automaton).

The arithmetic device, as its name indicates, serves for the performance of arithmetic operations. However the arithmetic unit is usually also used for the performance of other operations, logical for example. In this connection it would be more accurate to term the arithmetic device an operational device. However we shall not deviate from established tradition in the terminology of the AU.

Finally, the control unit combines and coordinates the operation of the all remaining devices of the universal program automaton, accomplishes the selection, decoding and organization of the instructions composing the program. In the modern universal program automata the control unit is constructed on the cyclic principle. The essence of this principle is that the operation of the automaton in time is broken down into natural intervals, termed the operating cycles, in the course of which there is repeated approximately the same sequence of elementary operations.

The determination of the beginning and end of the operating cycle is to a certain degree arbitrary, since their simultaneous shift in either direction is possible. We will assume that the operating cycle begins when in the control unit the command (instruction) subject to performance has already been transmitted. In the course of the cycle this command is performed: its operational part is used for the readying for performance of certain operations of both the control unit itself and of the arithmetic unit. The adress portion of the command is used for the excitation of the corresponding cells of the MU for the purpose of extracting from them or entering in them of certain information. In the multi-address systems the command for the decoding of the various addresses is accomplished sequentially. The operating cycle terminates with the extraction from the MU and the transmission to the CU of the code of the following instruction subject to performance.

We note that the CU not only transmits information to other units but also receives information from them: the command code from the MU, the result of the verification of the conditions defining the transfet to one or another of the two commands following after the conditional transfer command from the AU, and certain other signals from the AU. As we mentioned in the preceding section, in addition to the operational memory unit (OMU) the modern universal program automata are also equipped with an external memory unit (EMU) which is slower acting than, but at the same time of larger capacity in comparison with the OMU. The block diagram of a universal program automaton which defines the interaction (information exchange) between its basic units is shown in Fig. 18.

For the detailing of the block diagram let us consider in more detail the structure of the individual units composing it, and primarily the structure of the OMU, AU and CU. Essential component parts of all three devices mentioned are the so-called registers. A register is a memory cell which is intended for the storage of one information or command word. However, in contrast with the usual memory cells to which access is possible only after the accomplishment of quite complex preliminary commutation (switching), the registers are particularly accessible memory cells whose inputs and outputs are direclty connected to the circuits which transmit the information.

Depending on the method of functioning of these circuits in universal program automata (universal digital machines) are divided into two major classes: the series and parallel machines. In the parallel machines (automata) with the transmission of the code from register to register all the digits of this code are transmitted simultaneously, while in the series machines they are transferred sequentially, one after the other. It is clear that the parallel machines, other conditions being the same, will be faster acting then the series machines, although they require a larger number (equal to the number of digits in the machine codes of the words) of parallel channels for the transmission of the information between the registers, while in the series machines we can limit ourselves to one such channel.

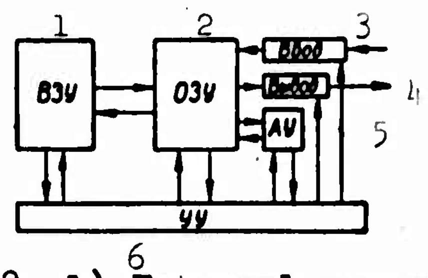

Fig. 18. 1) External memory unit; 2) operational memory unit; 3) input; 4) output; 5) arithmetic unit; 6) control unit.

The arithmetic unit of the modern universal electronic digital machines usually consists of three registers, one of which has the capability of summing the numerical codes transmitted to it and is therefore termed a summator. The numerical codes with which the arithmetic unit operates are numbers of differing signs, and the summation which we are discussing is understood as algebraic addition (with account for the signs). In the parallel machines the summation operation is usually performed in two elementary cycles of the machine. Here by elementary cycle we mean the interval of time between two sequential clock pulses applied in the AU. Most frequently the source of the clock pulses is the synchronizing generator which is common to the entire machine and is a part of its control unit. There are also other methods of organization of the elementary cycles which are used in the so-called asynchronous machines. These methods are described in detail in the handbooks on the electronic digital machines.

The operations performed in the various portions of the universal program automaton in the course of a single elementary cycle are customarily termed microoperations. More complex operations which are performed over several elementary cycles are realized with the aid of a set of microoperations, termed the microprogram of the given operation. The microprogram of the summation operation in the arithmetic units of the parallel machines usually consists of two microoperations: the microoperation of bit-organized addition and the microprogram with which there are realized the carries from some places to others which arise as a result of the place-by-place addition. In this case it is assumed that one of the addends was established on the summator ahead of time and the other on one of the AU registers.

We can, moreover, also construct the AU so that after the setting of the addends on the register and the summator the addition will be accomplished as a result of only a single microoperation — the transfer of the augend from the register into the summator. In the future we shall consider that the AU which we are discussing is constructed in just this way. Summators of such AU are custimarily termed accumulators, since they have the capability of accumulating the sum of any number of terms as the result of the sequential transmission to the summator of all the terms one after another.

Making use of the circumstance that the summator performs algebraic addition (with account for the signs of the addends) it is not difficult to organize on such a summator the operation of subtraction as well, by performing the transfer (from register to summator) or the code of the subtrahend as an ordinary addend but with the sign reversed. In the set of microoperations of the universal digital machines we therefore introduce the microoperations not only of conventional (direct) transfer of the number codes, but also the transfer of the code with its sign changed. We must also provide for the microoperations of register clearing, as a result of which on the cleared registers there must be established the number codes which are the representation of the number 0.

For the performance of the operations of multiplication and division with the natural method of coding the numbers, the described microoperations are not sufficient. Therefore, along with the microoperations already described in the set of the microoperations of the universal digital machines we also introduce the microoperations of left and right shift on the registers. With performance of the microoperation of left shift the number code $x_1\ x_2\ \ldots\ x_n$ set in the register is replaced by the code $x_2\ x_3\ \ldots\ x_n 0$, and with performance of the right shift microoperation – by the code $0\ x_1 x_2\ \ldots\ x_{n-1}$. The code sign (not specially designated here) retains its value. Here the code digits on the right usually represent the lower digits of the number and the digits on the left represent its higher digits. Therefore we also say the with a right shift the number code is shifted in the direction of the lower digits, and with a left shift – in the direction of the higher digits.

As the feedback signals transmitted from the AU to the CU we usually select the signals on the sign of the number code which is in the summator and on the digit of the lowest place of the number code which is in one of the AU registers, which we denote by the letter P_2.

This register is also termed the multiplier register. The second register of the AU does not have feedback connection with the CU. It is usually termed the multiplicand register and is denoted by the letter P_1.

As a rule, in the MU there are only two registers, one of which is termed the number register, and the other the address register. On the address register there is stored the address of the cell of the MU with which operation is to be performed (writing or reading of the code) and on the number register there is the number code being selected from the MU or being sent there for stroage. In addition, there are usually two other channels for the receipt of signals from the CU on the nature of the coming operation (writing or reading). The transmission of a pulse along one of these channels (write channel) leads to the code set in the address register being memorized (written) in the MU cell whose address coincides with the code set in the address register. The transmission of a pulse along the other channel (read channel) leads to the code from the MU cell whose address is set in the address register being transferred to the (previously cleared) number register.

The two described MU microoperations are termed respectively the microoperations of MU read-in and read-out. In addition to these microoperations, in the MU provision is made for the MU register clear microoperation and also the microoperation of the transfer of codes from the AU and MU registers into the number and address registers and the reverse transfers from the MU into the AU and CU. Speaking of the transfer of a code from register to register, we must always understand, if not stipulated otherwise, ordinary transfer, i.e., transfer of the code without change of its sign.

Let us analyze the structure of the CU, following the scheme of microprogramming control, first described by Wilkes and Stringer [75]. The microprogram control unit has in its composition two registers, termed respectively the command register and the microoperation register. The command register serves for the storage of the command (instruction) currently being performed. In accordance with the accepted structure of the instructions, the command register is subdivided into several registers — the operation register, the first address register, the second address register, etc. In the description of the microprograms it is sometimes convenient to work with the command register as a whole, and sometimes to break it up into the component parts.

The microoperation register, sometimes also termed the microcommand register, serves for the storage of the code of the microprogram (microcommand) instruction which is being performed at the given moment, i.e., the code denoting the ensemble of the microoperations being performed at a given moment.

In addition to the command and microoperation registers, in the universal program automata with the natural order of succession of instructions (see §1 of the present chapter) there is still another register, termed the command counter. With the application of a pulse to a special input of this register, there takes place an increase by unity of the number code set in it prior to this time. If the command counter were cleared ahead of time, then it will obviously perform a count of the pulses arriving at its input. This is where the register derives its name. The command counter serves for the storage of the command addresses. In the process of the performance of an instruction, not an instruction for conditional transfer, there is an increase of the command counter contents by one and the selection of a new instruction from the MU in accordance with the address thus obtained.

The increase of the contents of the command counter by unity is one of the microoperations of the CU. Among the other microoperations provided for in the CU we note the clearing of the registers, the transfer of codes from the MU number register into the command register, the transfer of codes from the CU address registers (first, second, etc.) into the MU address register, the transfer of a code from the command register into the summator (for accomplishment of the readdressing operation) and the transfer of the code from one of the CU address registers (usually from the third) into the command counter (with performance of the conditional transfer operation). For the performance of the logical operations which were mentioned in the preceding section, several new microoperations are added to the number of microoperations of the arithmetic unit.

As for the microprogram control unit itself, in the Wilkes and stringer scheme it includes, in addition to the microoperation register mentioned above, the so-called microoperation decoder and two diode matrices,* termed the A-matrix and the B-matrix. A simplified symbolic circuit of the microprogram control unit is shown in Fig. 19. On this figure the letters POn denote the operation register (a component part of the CU command register) and PMO denotes the microoperation register. The dots designate the points of connection of the diodes which connect the horizontal buses of the matrix with the vertical buses. The purpose of the diodes is to pass the pulses in the forward direction (from the horizontal buses to the vertical) and to prevent their passage in the reverse direction. With this condition the pulse applied to any horizontal bus D goes to those and only those vertical buses with which the given bus D is connected by the diodes. If the connection of the buses is accomplished indirectly, false paths for the passage of the pulses are possible which were not intended by

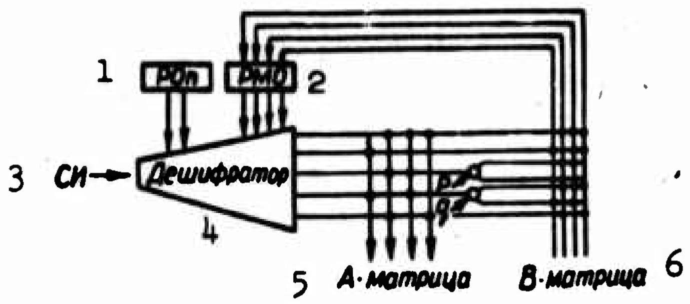

Fig. 19. 1) Operation register; 2) microoperation register; 3) synchronizing pulse; 4) decoder; 5) A-matrix; 6) B-matrix.

the designer. For example, with pulse application to the second-from-the-top horizontal bus of the A-matrix the pulse would appear not only on the first left vertical bus but also on the third from the left, passing to it through the second-from-the-bottom horizontal bus. The use of the diodes eliminates the possibility of the formation of such false paths, since the reverse transfer of pulses from the vertical buses to the horizontal is not possible.

The purpose of the decoder shown in Fig. 19 is the transmission of the successive pulse SI of the synchronizing generator applied to its input to precisely the one of the horizontal buses of the A-matrix which is uniquely determined by the codes set at the considered moment of time in the operation and microoperation registers. Some of the horizontal bus of the B-matrix and some go into two such buses. In the latter case the transfer of the pulse from the horizontal buses of the B-matrix is determined by the feedback signal (by the signal p or q in Fig. 19) coming from the AU.

Transferring to the vertical buses of the B-matrix (determined by the method of connection of the diodes) the pulses enter the microoperation register, altering the code previously set there. Therefore the following pusle of the SI, passing through the decoder, will go to a new horizontal bus. By connecting the vertical buses of the A-matrix to the corresponding units of the machine so that the transmission of a pulse along each of the vertical buses leads to the performance of precisely one microoperation, we obtain the possibility by using this process of accomplishing any finite sequence of microoperations (microprogram), in this case combining several of the microoperations into one microcommand.

By writing the microprograms for the various operations which the machine must perform, we determine the method of connection of the diodes in the A-matrix and in the B-matrix and consequently the structure of the entire microprogram control unit. With the performance of the microprogram of any machine operation the contents of the operation register remains unchanged, while the contents of the microoperation register varies with every new elementary cycle. Only at the very end of the performance of the operation, after the selection of the new instruction from the memory, does the contents of the operation register change, after which there begins the performance of the microprogram of the succeeding operational cycle of the machine.

Since the structure of the control unit and even of the entire machine is to a considerable degree determined by the selection of the operations and the microprograms for them, let us consider concrete examples of microprograms for the most common machine operations. Here for definiteness we shall consider that the machine under discussion is a three-address parallel universal digital machine with natural order of succession of the commands. We use the letter $\underline{p}$ to denote the number sign function (± 1) of the number located in the summator, and $\underline{q}$ to denote the value of the lowest place of the number in the multiplier register P_2 (it is assumed that the machine operates with n-place proper fractions in the binary notation system).

<u>Addition Microprogram</u>

1. Clear AU and MU registers.

2. Transfer of the code from the CU first address register into the MU address register.

3. MU read-in.

4. Transfer code from MU number register into AU summator.

5. Clear MU registers.

6. Transfer code from CU second address register into MU address register.

7. MU read-in.

8. Transfer of code from MU number register into AU summator (most frequently via the AU register P_2).

9. Clearing of MU registers.

10. Transfer of code from AU summator into MU number register.

11. Transfer of code from CU third address register into MU address register.

12. MU read-in.

13. Clearing of MU registers.

14. Increase of command counter content by one.

15. Transfer of code from command counter into MU address register.

16. MU read-out.

17. Clearing of command register.

18. Transfer code from MU number register into command register.

Certain of the microoperations making up the described microprogram can be combined in time, as a result of which the time for the operating cycle can be reduced and the speed of the machine can be increased. Examples of such combined microoperations might be the microoperations 16 and 17 or microoperations 10 and 11.

Microprogram for Multiplication by Positive Number

1. Clear AU and MU registers.

2. Transfer code from CU first address register into the MU address register.

3. MU read-in.

4. Transfer code from MU number register into AU register P_1.

5. Clear MU registers.

6. Transfer code from CU second address register into MU address register.

7. MU read-in.

8. Transfer code from MU number register into AU register P_2.

9. (1) Transfer code from P_1 register into summator if $q = 1$, and go to the following microoperation without number transfer if $q = 0$.

10. (1) Right shift on summator.

11. (1) Right shift on P_2 register.

9. (2) Same as 9(1).

10. (2) Same as 10(1).

11. (2) Same as 11(1).

. . . .

. . . .

9. (n) Same as 9(1).

10. (n) Same as 10(1).

11. (n) Same as 11(1).

12. Clear MU register.

13. Transfer code from CU third address register into MU address register.

14. Transfer code from summator into MU number register.

15. MU write-in.

16. Clear MU registers.

17. Add one of content of command counter.

18. Transfer code from command counter into MU address register.

19. MU read-in, clear command register.

20. Transfer code from MU number register into command register.

In the multiplication microprogram, in contrast with the addition microprogram, considerable use is made of the possibility of branching in the order of succession of the microoperations as a function of the feedback signal q coming from the AU. It is not difficult to see that the sequential performance of microoperations 9, 10, 11 is equivalent to the performance of the usual, well known multiplication algorithm with roundoff for the binary notation system.

Actually, in the described technique the AU summator serves for the storage of the sums of the partial products of the multiplicand by the individual digits of the multiplier. This storage is accomplished with an accuracy to the lower digits which are dropped in the right shift of the summator contents. Each time the multiplicand is added or not added to the partial sum stored in the summator, depending on whether the lowest digit of the right-shifted multiplier is equal to one or zero. It is clear that this procedure together with the preliminary shifts on the summator and the P_2 register denotes the addition of the product of the multiplicand by the next (right) digit of the multiplier in the previous sum of the analogous (partial) products, as is required in the conventional multiplication algorithm. Roundoff to the number of significant digits contained in the cofactors is accomplished as a result of the shifts on the summator which lead to the elimination of the lowest digits of the product. In the case of multiplication not only by positive, but also by negative numbers an additional microoperation for the formation of the sign of the product is introduced into the microprogram.

<u>Microprogram for Conditional Transfer on Inequality</u>

1. Clear AU and MU argisters.

2. Transfer code from CU first address register into MU address register.

3. MU read-in.

4. Transfer code from MU number register into AU summator.

5. Clear MU registers.

6. Transfer code from CU second address register into MU address register.

7. MU read-in.

8. Transfer code from MU number register into AU register.

9. Transfer code with sign change from register into summator.

10. If the summator content $s > 0$, then add one to content of command counter, if $s \leq 0$ then transfer code from CU third address register into command counter.

11. Clear MU registers.

12. Transfer code from command counter into MU address register.

13. MU read-in.

14. Clear command register.

15. Transfer code from MU number register into command register.

This microprogram accomplishes the comparison of the number A_1 written from the first address of the command with the number A_2 written from its second address. If it is found that $A_1 > A_2$ control is transferred to the next command in order. If, however, $A_1 \leq A_2$ the following command is taken from the third address of the current command. It is not difficult to see that from the two operations of conditional transfer with resprct to inequality, by switching the places of the numbers A_1 and A_2 we can form the operation of conditional transfer with respect to the exact coincidence of the words described in the preceding section.

The described basic principles of the organization of the algorithmic process in the universal electronic digital machines gives a general idea of the so-called <u>block structure</u> of such machines. In the real design of the electric programming automata the stage of the block synthesis is only the starting point for the development of particular circuit solutions. The selection of these solutions is made on the basis of the theory of automata and the theory of combination circuits presented in Chapters 1 and 3.

§3. THE CONCEPT OF PROGRAMMING

Programming is the writing of a particular algorithm in the form of a finite sequence of instructions (commands) for the universal program automaton. Such a sequence is termed the operating program of the given automaton. Entered into the automaton together with the initial data (input word of the algorithm), it forces the automaton to perform the operation of the algorithm in question, i.e. to transform the given input word (input information) into the corresponding output word (output information). The universality of the set of operations of the modern program automata provides the possibility of programming any algorithm with the condition that we neglect the limitations imposed by the finiteness of the volume of the automaton memory.

In order to understand the essence of the problems posed in programming, let us consider first some simple particular example. Let us assume that we are required to calculate a sum of the form $\sum_{k=1}^{m} \frac{1}{k^3}$, where m is some fixed natural number. We shall compile the program for the solution of this problem for a three-address universal digital machine with natural order of succession of commands, whose set of operations includes all four arithmetic operations (addition, subtraction multiplication and division), the operation of conditional transfer with respect to exact word coincidence and the operation of machine stop.

Let us assume for simplicity that there is no need for input and output of information in the machine MU. In other words, the initial information and the program which will be compiled are assumed to have been entered in the proper memory cells and the problem solution is obtained in the MU cells (in a single cell in the considered case) assigned for this purpose after stopping the machine.

For the solution of the posed problem we introduce the following notations:

$$x_k = k;$$
$$y_k = x_k^2 = k^2;$$
$$z_k = \frac{1}{y_k} = \frac{1}{k^2};$$
$$s_k = z_1 + z_2 + \ldots + z_k = s_{k-1} + z_k.$$

We shall assume that for the storage of the quantities x_k, y_k, z_k and s_k there are assigned some four memory cells termed the working cells. It is natural to break the process of the computation of the desired sum s_k down into m completely identical steps, in each of which we compute the corresponding value of the partial sum s_k ($k = 1, 2, \ldots, \ldots, m$). The computations are initiated with the value $k = 1$ and $s_0 = 0$ and can be written in the form of the following scheme:

1) $y_k = x_k^2$;
2) $z_k = \frac{1}{y_k}$;
3) $s_k = s_{k-1} + z_k$;
4) $x_{k+1} = x_k + 1$;
5) if $x_{k+1} = m + 1$ then go to the following instruction; if $x_{k+1} \neq m + 1$ then return to instruction 1;
6) stop.

This scheme is actually already the desired program, in whose instructions the actual addresses of the working cells are replaced by the symbolic designations x_k, y_k, z_k, s_k termed symbolic addresses. By fixing the actual addresses of these cells, and also of the cells containing the constants 1 and $m + 1$ which figure in the program in question, it is not difficult to go from the program with symbolic addresses to the actual program in the three-address instructions. In this case we shall assume that the conditional transfer instruction provides for natural succession of instructions with coincidence of the compared words (located in the first and second addresses of the conditional transfer instruction) and transfers control to the third address with noncoincidence of the compared words.

Let the addresses of the working cells x_k, y_k, z_k and s_k be respectively 1, 2, 3 and 4; let the addresses of the cells containing the constants 1 and m + 1 be 5 and 6; and let the address of the cell containing the first program instruction be 7. In this case the desired program (in actural addresses) is written in the form of the follwoing sequence of instructions:

1) multiplication 1, 1, 2;
2) division 5, 2, 3;
3) addition 4, 3, 4;
4) addition 1, 5, 1;
5) conditional transfer 1, 6, 7;
6) stop.

In the cell with the address 1 there must initially be placed a number equal to one, and in the cell with the address 4 there must be a number equal to zero. The initial filling of the working cells with the addresses 2 and 3 is evidently unimportant, since the required filling of these cells is performed by the first and second instructions of the program. We recall that the machine memory is presumed to be constructed so that with the writing or any word in any cell the previous content of this cell is erased. After stopping of the machine (performed by the sixth instruction) the sought value of the sum s_m is obtained in the fourth memory cell.

It is not difficult to see that in this program the cell with the address 2 can be used not only for storing the quantity y_k, but also for the storage of the quantity z_k. Actually, the quantity y_k is used exclusively only for the calculation of the value of the quantity z_k, therefore after calculating z_k the value found can be sent to the cell where the quantity y_k was previously stored without danger of interfering with the correctness of the following calculations. There exist general techniques which permit automating such a process of economizing of the working cells.

As the second example let us consider the programming of the problem of the calculation of the scalar product of two n-dimensional vectors $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_n)$ i.e., the computation of the sum s of the paired products of their corresponding components: $s = a_1b_1 + a_2b_2 + \ldots + a_nb_n$. Let us assume that the components of the vector A are arranged in the cells with the addresses $a + 1, a + 2, \ldots, a + n$, and the components of the vector B are in the cells with the addresses $b + 1, b + 2, \ldots, b + n$, where a and b are certain fixed natural numbers, chosen so that the arrays of the cells assigned for the storage of the components of the vectors A and B are disjoint.

For the sake of variety let us compile the program for the calculation of the scalar product with application to a single-address machine with natural order of succession of commands. In this case the instructions which realize the arithmetic operations are understood so that the corresponding operation is performed with a pair of numbers, the first of which is in the AU summator, and the second is in the cell whose address is indicated in the instruction. The result of the operation remains in the summator. For the performance of the arithmetic operations with such instructions it is also necessary to have instructions which accomplish the exchange of commands between the AU summator and the MU cell whose address is indicated in the corresponding instruction.

The conditional transfer command can be performed in various ways. Let us assume that there takes place a conditional transfer for zero in the summator: if in the AU summator with the performance of a command for conditional transfer there appears written the number 0, then control is transferred to the next command in order, otherwise the selection of the following command is made from the address indicated in the conditional transfer command. It is assumed for simplicity that the ith command of the program will be stored in the cell with the address i (i = 1, 2, ...). Assigning for the storage of the readdress constant (number equal to one) the cell with the address c, for the storage of the number n (dimension of the vectors A and B — the cell with the address a, and for the storage of the partial sum $s_k = a_1b_1 + a_2b_2 + \dots + a_k b_k$ — the cell with the address s, we arrive at the following program:

1) transfer to summator from MU, a + 1;

2) multiply, b + 1;

3) add, s;

4) transfer from summator to MU, s;

5) transfer to summator, 1;

6) add, c;

7) transfer from summator to MU, 1;

8) transfer to summator, 2;

9) add, c;

10) transfer from summator to MU, 2;

11) transfer to summator, t;

12) add, c;

13) transfer from summator to MU, t;

14) transfer to summator, a;

15) subtract, t;

16) conditional zero transfer to summator, 1;

17) stop.

The cell with the address t which appears in the constructed program is the so-called program counter of the number of cycles. Instructions 11, 12 and 13 perform the increase of the content of this cell by one. If in the beginning of the computation there was a number equal to zero in the cell t, after the performance of n cycles (repetitions of the group of instructions 1-13) there will be a number stored in this cell equal to n. Then as a result of the subtraction performed by instructions 14 and 15, zero will be obtained in the summator and the subsequent conditional transfer command will transfer control to command 17, which stops the machine.

Up to this moment the conditional transfer command will transfer control to the first command, as a result of which the computation cycle will be repeated. However, this will not be a literal repetition, since with the aid of commands 5, 6, 7 and 8, 9, 10 there is accomplished an increase of the address of the first and second commands of the program by unity. Therefore commands 1 and 2 will lead to the formation of the product of a new pair of components $a_k b_k$ of the vectors A and B, commands 3 and 4 will lead to the computation and the storage in the cell s of the new value of the partial sum $s_k = s_{k-1} + a_k b_k$.

For a proper understanding of the described program it is necessary to note that the operation of transfer into the summator of any code assumes the preliminary clearing of the summator and after the transfer of the code from the summator to the MU the summator is also automatically cleared. In addition, it is necessary that in the beginning of the operation there be a number equal to zero written in the cell t.

The described method for performing the address substitution (change of the command addresses) is not convenient when the codes of the commands themselves are subject to change. First, it leads to extension of the program (particularly noticeable in the case of the s single-address machines), and second, and this is most important, as a result of its use the initially given program is altered and is not suitable for further use without preliminary restoration of the original values of the address portions of the commands. This restoration introduces further complications in the program and requires additional MU cells for the storage of the original addresses. Therefore for the majority of the modern universal digital machines we prefer another method of readdressing, associated with the use of the so-called <u>address modification registers</u>, or <u>index-registers</u>.

The index registers are a part of the control unit of the universal program automaton and have the property that in the process of the performance of any command the contents of a particular one of them is automatically added to those command addresses which are equipped with a special <u>label</u> corresponding to this index register.

With the use of a single index register I in the three-address command system, the program for the computation of the scalar product of the vectors can be written with only five commands in all (designations of the cell addresses are the same as in the preceding program):

1) add 1 to the content of the index register I;
2) multiply, a(I), b(I), <u>p</u>;
3) add, <u>p</u> <u>s</u> <u>s</u>;
4) conditional transfer, I, <u>a</u>, 1;
5) stop.

In this program use is made of conditional transfer on exact coincidence of codes in the index register I and in the cell <u>a</u> (where the dimension $\underline{n}$ of the vectors A and B is written). In the case of coincidence of these codes control is transferred to the sequentially following (fifth) command, and in the case of noncoincidence control is transferred to the first command of the program.

In the course of the entire time of operation the program retains its initial form, only the content of the index register I changes. In case of necessity, there may be included in the program a special command for the clearing of the index register, setting it to zero.

In the programming of more complex algorithms, for example the algorithm for the multiplication of a vector by a matrix, the need arises to use several index registers for the storage of the readdressing constants which are changed by the various steps. Let us consider as an example the multiplication of the n-dimensional vector $B = (b_1, b_2, \ldots, b_n)$ by the matrix A of nth order with the elements a_{ik} $(1 \leq i \leq n,\ 1 \leq k \leq n)$.

Let us assume that the sequential components of the vector B are located in the memory cells with the addresses $b + 1, b + 2, \ldots, b + n$ and the elements a_{ik} of the matrix A are in the memory cells with the addresses $a + (k - 1)n + i$ $(i, k = 1, 2, \ldots, n)$. The components of the vector $C = BA$ are located in the cells with the addresses $c + 1, c + 2, \ldots, c + n$ (there were initially numbers equal to zero in these cells). We use the cell with the address $\underline{t}$ as the working cell for storage of the intermediate results (its initial content is not important to us). Finally, the cells with the addresses 1, 2, ... are used for the storage of the sequential instructions which constitute the sought program.

Let us introduce the three index registers I_1, I_2, I_3, which must be cleared prior to initiation of operation, and let us place in the cell with the address $\underline{d}$ the number $\underline{n}$, equal to the dimension of the vector B and the order of the matrix A. With the aid of the introduced notations the desired program for the multiplication of the vector by the matrix is written in the following form:

1) add 1 to the content of the index register I_3;
2) add 1 to the content of the index register I_1;
3) add 1 to the content of the index register I_2;
4) multiply, $b(I_1)$, $a(I_2)$, $\underline{t}$;
5) add, c (I_3), $\underline{t}$, $c(I_3)$;
6) conditional transfer, I_3, $\underline{d}$, 2;
7) clear index register, I_1;
8) conditional transfer, I_3, $\underline{d}$, 1;
9) stop.

With further complication of the algorithms the difficulties of the programming increase more and more. In this connection there naturally arises the thought of looking for more economical methods of writing the information on the algorithm and the application of the most universal program automaton for the automatic translation of such forms into the actual operational programs. This idea constitutes the basis for automatic programming with the aid of the so-called <u>universal programming programs</u> (translators).

The universal programming program is an algorithm programmed for a particular universal digital machine for the translation of the statement of any algorithm in a particular <u>formal algorithmic language</u> into the instruction language of the given machine. As the formal algorithmic language in question here we can, of course, select any of the languages described in Chapter 1, for example the language of the normal algorithm schemes. However such a choice would not facilitate, but rather would complicate the solution of the programming program, since the statement of the algorithm in any of the abstract algorithmic schemes of the first chapter is, as a rule, a considerably more difficult problem than programming in the language of the instructions of the modern universal digital machines. To convince ourselves of this it is sufficient to consider the case of the addition operation, which is performed in the universal digital machines with the aid of one instruction while, for example, with the use of the normal algorithms it is realized with the aid of the quite complexly written scheme which contains many elementary substitutions.

Therefore attempts have been made to develop those universal algorithmic languages which would retain the basic properties of the language of the modern universal digital machines but which would permit a simpler and more easily read statement of the algorithms encountered in practice in comparison with the direct programming in the "machine" languages. Among the languages of this sort we note, for example, the Fortran language (U.S.), the Polish algorithmic language SAKO, the address language (Kiev, USSR), and others.

The creation of practical algorithmic languages is important not only because such languages facilitate programming, but also because a sufficiently well developed practical algorithmic language can become a generally accepted and generally understood language for the writing of various algorithms. Thus, the ALGOL-60 language which was developed by a group of European and American scientists has at the present received wide international acceptance. A detailed description of this language is given in the following chapter, here we shall consider certain techniques which facilitate direct programming in machine languages.

The first technique, already considered above, is the use in the initial stage of the programming of symbolic addresses in place of the actual (numerical) addresses. The later assignment of the actual values to the introduced symbolic addresses and the economy of the working cells are a purely technical operation and are easily subjected to automaton. In spite of its simplicity, the method of symbolic addresses permits significant simplification of the programming of complex problems and, what is most important, considerably reduces the number of errors.

A second method which can be used to significantly simplify direct programming is the inclusion of previously constructed simpler programs in the more complex programs. The programs specially adapted for inclusion in the more complex programs are usually termed <u>subprograms</u>. By accumulating a <u>library of subprograms</u>, the programmer can in many cases reduce the direct programming to a combination of a small number of avialable subprograms. To facilitate the combining of several subprograms into a single program, there have been worked out special techniques which make it possible to avoid introducing changes in the subprograms when including them in quite diverse programs. The difficulty lies in the fact that the last instruction of the subprogram must transfer control to some instruction of the basic program, which changes from program to program and is not known to the complier of the subprogram.

We can overcome this difficulty by sending at the moment of transfer to the subprogram the address of the instruction to which the machine must go after completion of the subprogram into a special memory cell which is termed the <u>return register</u>. In this case the subprogram must be terminated by a special instruction "transfer on the return register," which extracts the next command from the cell whose address is stored in this register. We note that for the use of any given cell of the machine operational memory as the return register it is advisable to introduce referral to the memory using the so-called <u>second rank address</u>. With this referral the selection from the memory or the writing into the memory are accomplished not by the addresses indicated in the command being executed, but by the addresses which are stored in the memory cells whose addresses are indicated in this command. With the use of subprograms which are included in other subprograms (and not in the basic program) we must make use of several return registers or second rank address transfers.

Such use of some subprograms within others creates the basis for the multistage organization of systems of standard programs. The rationality of such organization is determined by the degree of economy of the arrangement of the library of standard programs in the particular memory devices. This question becomes particularly important with the scheme of realization of various sorts of standard subprograms which enrich the set of operations performed by the machine. In this case there is achieved a major economy of the work of the programmers, who find it possible to use a large subprograms, assigning to each of them only a single machine instruction. Such multistage organization of the control is realized in the "Promin'" computer of the Institute of Cybernetics of the Academy of Sciences of the Ukrainian SSR (Kiev).

Moreover, even in the absence of the schematic realization a sufficiently extensive library of standard subprograms significantly facilitates the programming, since a considerable portion of the new programs being compiled will, as a rule, be made up of previously programmed standard portions available in the library.

We note that in compiling a library of standard subprograms an attempt is made to provide a quite high degree of generality of the problems being solved. For example, the standard subprogram for the multiplication of matrices is compiled for the multiplication of matrices of any order rather than for only some one order. Other subprograms are constructed similarly.

Also of significant assis ance in direct programming is the preliminary writing of the program .n simplified form, usually termed the program block diagram, with subsequent programming of each individual block. A convenient method for writing program block diagrams is the operator method of writing program diagrams proposed by Lyapunov.

In the use of this method groups of program commands of a single type which follow one another (for example, commands which realize the arithmetic operations) are combined into the so-called operators. The most widely used are the arithmetic operators and the readdressing operators (which change the content of the index registers). We label the arithmetic operators with the letter A, the logical operators with the letter P, the readdressing operators with the letter I, and the stop operator with the letter F. In addition, the operators are numbered with the use of special indices in the order in which they occur in the program.

Combining a group of arithmetic operations, every arithmetic operator is a coded designation for the operation of computation using a particular, frequently quite complex, formula. The logical operator makes a verification of the logical conditions on the basis of which particular conditional transfers are performed (control transfers in the program which violate the natural order of succession of commands). A vertical bar is placed after the logical operator; above and below this bar there are indicated the numbers of the operators to which control is transferred in the case when the logical condition is satisfied, and correspondingly in the case when it is not satisfied. Absence of a number below or above the bar indicates that in the corresponding case conteol is transferred to the operator standing directly to the right of the bar..

With the aid of the described symbolism the operator diagram of the last of the programs which we considered can be written as

$$I_1 I_2 A_3 P_4 \underset{2}{|}\, I_5 P_6 \,\Big|\, F_7.$$

Here the readdress operator I_1 corresponds to the first instruction of the program, and the operator I_2 corresponds to the two following instructions. The arithmetic operator A_3 combines the fourth and fifth instructions, and the remaining operators include one instruction each.

To facilitate reading of the operator diagrams the bars which designate the conditional transfers can be supplied with horizontal lines above and below, directed to the left. In this case there is also placed ahead of the operator to which control is transferred a bar with a line directed to the right and labelled with the same number as the corresponding bar of the conditional transfer operator. Then a group of operators which composes a cycle which is repeated several times as a result of the conditional transfers is framed on both sides by sort of "brackets" which facilitate the search for such cycles.

Using these notations, the operator diagram described above can be rewritten

$$\underset{1}{\llcorner} I_1 \underset{2}{\llcorner} I_2 A_3 P_4 \underset{2}{\lrcorner} I_5 P_6 \underset{1}{\lrcorner} F_7.$$

By supplying the operator diagram of the program with the description of each of the operators occurring in it (other than the stop operator) we can after the compilation of such a diagram turn to the individual, sequential programming of these operators with subsequent combining of the individual pices of the program thus compiled into a single whole, These operations are to a considerable degree routine work and can be relatively easily automated with the aid of any universal program automaton.

We note that for the description of the arithmetic and logical operators we can make use of the conventional arithmetic or logical formulas. Having available the special programs for the automatic translation of such formulas into the machine instruction language and combining them with standard library subprograms, we find the possibility of presenting to the machine (universal program automaton) the task in the same from in which it is presented to the skilled human computer. This method actually combines the method of the standard subprograms with the method using (to a certain extent) the universal programming programs. Therefore it is natural to term it the specialized programming program method or the programming program library method [24]. Here the specialization consists in the fact that the corresponding library is oriented to a certain class of typical problems, permitting actually the complete elimination of programming and limiting oneself to communicating to the machine only the conditions of the problem which must be solved.

§4. THE UNIVERSAL ALGORITHMIC LANGUAGE ALGOL-60

The international algorithmic language ALGOL-60, which for brevity we shall term simply ALGOL, is a means for the quite simple, precise and clear writing of computational algorithms. Being a universal algorithmic language, it is suitable, of course, for the writing of any (not necessarily computational) algorithms, however in the case of the processing of literal rather than numerical information the simplicity and the clarity of the corresponding "algol" writing is to a considerable degree lost. While the programming of the computational algorithms by ALGOL is a far simpler problem than the direct programming for the modern universal electronic digital machines, the programming of problems on the processing of literal information by ALGOL is only slightly simpler than using the "machine" languages.

The basic symbols used in the construction of the ALGOL language are the Latin letters (26 capital and 26 lower case letters), the Arabic numerals (from zero to nine inclusive), the logical values "true" and "false," and also the operation symbols, separator symbols and brackets (the last three types of symbols are termed limiters). There are also a certain number of service words, for which words of the English language are usually used. It is customary to write these words in bold face type.

For the notations of the numbers use is made of the decimal notation system, with the whole part being separated from the fractional part by a point (and not by a comma). The plus sign ahead of positive numbers and the zero symbol in the designation of the whole part of a proper fraction can be dropped. For the designation of a decimal exponent (number of tens in an integral power) we make use of a special symbol – a ten dropped below the basic line (it is usually printed in bold face type). The numbers used in ALGOL are divided into two types: integer and real. The integer type includes only the whole numbers (with or without sign) which do not contain in their writing a symbol of a decimal exponent or a decimal point; all the remaining numbers belong to the real type (here the number 3.0 is real but not an integer).

Examples of the integer type numbers are: 0, + 275, – 0634, + 0, – 2. Examples of the real type numbers are: + $5.340_{10}8$ (i.e., the number $5.34 \cdot 10^8$), – .063 (the number – 0.063), – $.37_{10}$ – 32 (the number $-0.37 \cdot 10^{-32}$), + $_{10}5$ (the number 10^5), etc.

The introduction of particular quantities in ALGOL (in writing of specific programs) is accompanied by their preliminary description. The subsequent concrete representations of these quantities must be interpreted in accordance with the indicated descriptions. If, for example, some quantity $\underline{x}$ was described by the term integer, and then its value was introduced equal, say to 23.4, then this value must be mentally rounded off to the nearest integer (23 in this case). The value of a quantity of the integer type, equal to 23.5, is rounded off to 24, and not to 23 (to the nearest larger integer).* We note also that if the quantity $\underline{x}$ described as real, nothing prevents it taking also integral values, however in subsequent operations with the quantity $\underline{x}$ we proceed just as with any real quantity without performing rounding off to the nearest integer.

For the designation of various kinds of quantities (constants and variables) in ALGOL use is made of the so-called <u>identifiers</u>. Any finite sequence of letters (Latin) and decimal digits, of necessity beginning with a letter (and not with a digit), can serve as an identifier. Examples of identifiers might be a7LO, x, ga, aPg, TOWW etc. At the same time the expressions 7x, bab or ab + $\underline{x}$ cannot serve as identifiers. The use for designation of quantities not only of the letters, but also of words, i.e., sequences of letters (possibly, meaningless) makes the supply of identifiers potentially unlimited, which is quite important from the point of view of the possibility of the representation of any algorithms, no matter how complex. The possibility of the designation of a quantity by its natural name also presents obvious conveniences, for example: force, current, etc. At the same time there is one inconvenience with which we must contend in the future: in the construction of arithmetic expressions from the identifiers the multiplication sign cannot be dropped (as is usually done in algebra), since the expression ab + xy will be understood in ALGOL as the sum of two quantities designated by ab and xy and not as the sum of the paired products of the quantities $\underline{a}$, $\underline{b}$ and $\underline{x}$, $\underline{y}$.

In addition to the quantities which take numerical values (of the integer and real type) in ALGOL use is made of Boolean quantities, taking only two values — "true" and "false." The Boolean quantities are designated by identifiers in just the same way as the numerical quantities; in the descriptions they are assign‿‿ to the Boolean type.

Similar quantities, for example the components of any vector or the elements of a particular matrix, are usually denoted by identifiers with one or several indices. The indices are written after the identifier and are enclosed in square brackets. Different indices are separated from one another by commas. Whole numbers (not just positive ones), variables and any arithmetic expressions which are always of the integer type, can be used as indices.

Examples of writing of variables with indices are: A[1, – 2], ps [i] bA8[i, j, 1].

Variables with indices which vary within certain limits constitute the so-called arrays. The array description in ALGOL is preceded by the English word array, before which there is placed the name of the type (integer, real, Boolean) of the variables composing the array (if the name of the type of variables in the array is not indicated, it is considered that they are of the real type). In the description of the array, after the array identifier in index (square) brackets there is written the so-called list of bound pairs. Each bound pair consists of two arithmetic expressions (or numbers) separated by a colon. The first of these expressions is the lower bound (smallest possible value) of the corresponding index, and the second is its upper bound (highest possible value of the index). It is assumed that the index can run through all the integral values included between the lower and upper bounds, and in the case when the upper bound is less than the lower, the corresponding array is considered indeterminate.

Examples of the description of arrays: **real array** *x*[1:*n*, 0:*m*], **Boolean array** *g*4[0:5], **integer array** *N*[— 7:1, *i*:*j*, 3:3].

The number of different indices characterizing an array is termed the dimension of this array. In the examples just presented the array x has a dimension of 2, the array g^4 has a dimension of 1. As for the array N, formally its dimension is equal to 3, however, since the last index can take only one single value (equal to 3) actually the value of the dimension of this array reduces to 2.

We note that in the descriptions of the variables or arrays of the same type the name of the type can be written only once, and the corresponding indetifiers are separated from one another by commas. In the case of arrays with the same bounds the index brackets with the corresponding list of bound pairs can be written out only once – after the last identifier of an array with these bounds. For example, real a, bx7 or integer **array** *A,B,Dd*[1:2, l:*k*] . The first description describes three variables which take real values, and the second describes three two-dimensional arrays with the same bounds which are composed of integral quantities.

For the separation of the described variables or arrays of different types use is made of a semicolon. For example, **real** *x,y*; **Boolean** *A,B,C*; **array** *px*[1:2, *l*:*k*]; **integer array** *N*[—1:0, 5:10], *Q*[2:4] .

Arbitrary arithmetic expressions, which play a large role in the construction of the ALGOL language, can serve as the index bounds in the arrays. The arithmetic expressions are constructed from numerals and variables with the aid of the six arithmetic operations – addition (denoted by the + sign), subtraction (denoted by the sign –), multiplication (denoted by the sign ×), division (denoted by the slant line symbol /), integral division (denoted by the sign –) and raising to a power (denoted by the symbol ↑).

The integral quotient a-b is the whole part (rounded off in the direction of reducing the modulus to the nearest integer) of the conventional quotient a/b. This operation is applied only to quantities of the integer type, so that the expression 3•0-5•0 is to be considered indeterminate, (while the expression 3-5 is determinate and equal to zero). The operation of raising to a power $\underline{a} \uparrow \underline{b}$ ($\underline{a}$ to the power $\underline{b}$) with positive $\underline{a}$ is determinate for all quantities $\underline{b}$ of the real and integer types, while for negative $\underline{a}$ it is defined only for the cases when the quantity $\underline{b}$ is of the integer type.

Other conditions being the same, in the arithmetic expression there must first be performed the operation of raising to a power, then the operations of multiplication and division (conventional and integral), then the operations of addition and subtraction. Like operations (multiplication and division or addition and subtraction) are performed in the conventional order – from left to right. When it is necessary to perform operations in a different order use is made of round brackets. With raising to a power $\underline{a} \uparrow \underline{b}$, the expressions $\underline{a}$ and $\underline{b}$ must as a rule be enclosed in brackets. Exceptions are permitted only in the case when the corresponding (not enclosed in brackets) quantity is an unsigned number, a variable (with or without indices), or a function (see below).

Examples of the arithmetic expressions are the expressions $\underline{x} \uparrow 2$ (equal to x^2), $3 \uparrow n \uparrow k$ (equal to $(3^n)^k$), $ab \times AB + pr \uparrow (-q)$, $(x7 + A9) \uparrow (-2)$ etc.

Along with the variables represented by the usual identifiers or by array identifiers, in the construction of the arithmetic expressions use is also made of the so-called functions. Every function in ALGOL is designated by the assignment to the function of an identifier after which there is placed in round brackets the so-called list of actual parameters, i.e., in other words, the arguments of this function. The actual parameters can be any expressions (arithmetic or Boolean) and also the array identifiers and certain other forms of identifiers which are defined below. The parameters are spearated from one another either by commas or by a line of the form:) any commentary: (. A commentary is the name given to a clarification of the meaning of the actual parameters, which we shall usually give in the Russian language. In the translation from the ALGOL language to the language of a particular computer this commentary is simply discarded.

Examples of the functions might be $f(x)$, $\Pi(x, y \pm a)$ $AB\,(k \uparrow 1.5)$ force: (p) acceleration: (a). The first of these functions is a single-place function (i.e., it depends on one actual parameter), the second function is two-place, and the third is three-place (it depends on the actual parameters k ↑ 1.5, p and a).

In the descriptions the functions are usually termed procedures with an indication of the type of quantity it defines (integer, real or Boolean). For such functions as sine, logarithm and others, we retain the commonly accepted identifiers sin, ln, etc. We establish the identifier sqrt for the designation of the square root, and the identifier abs for the designation of the absolute magnitude.

The descriptions of the functions include in themselves headings of the form real procedure *sin* (*x*), **integer procedure** *abs*(*n*); **Boolean procedure** *A*(*a*,*b*). In the descriptions use is made of the so-called formal parameters as the function arguments, i.e., certain identifiers which in the subsequent use of the function can be replaced by any actual parameters, i.e., variables, expressions (arithmetic, Boolean, designational), identifiers of arrays of procedures or swithces, and also the so-called lines.

The expressions constructed from numbers, variables of the real and integer (with or without indices) and functions (integral or real) with the aid of the arithmetic operations are termed simple arithmetic expressions. In order to construct more complex arithmetic expressions it is necessary to become acquainted with the so-called Boolean expressions.

Boolean expressions are constructed from the logical values ("true" and "false"), variables and functions (procedures) of the Boolean type and the so-called relations, which are two arithmetic expressions $\underset{\bullet}{A}$ and $\underset{\bullet}{B}$ connected with one another by the equality or inequality signs: $\underset{\bullet}{A} = \underset{\bullet}{B}$, $\underset{\bullet}{A} \neq \underset{\bullet}{B}$, $\underset{\bullet}{A} > \underset{\bullet}{B}$, $\underset{\bullet}{A} \geq \underset{\bullet}{B}$, $\underset{\bullet}{A} < \underset{\bullet}{B}$, $\underset{\bullet}{A} \leq \underset{\bullet}{B}$. As the operations for the construction of the Boolean expressions use is made of the logical operations described in Chapter 2: equivalence ($\equiv$), implication ($\supset$), disjunction ($\vee$), conjunction ($\wedge$) and negation ($\neg$). The proirity of the logical operations with respect to one another and the method of use of the brackets (only the round in the present case) are retained the same as in Chapter 2. We only need add that the arithmetic operations (expressions) are considered to take precedence over all the relation operations, and the latter have precedence over all the logical operations, so that the expression $a+b>c\times d\supset x\wedge y\vee z$ must be understood as $((a+b)>(c\times d))\supset((x\wedge y)\vee z)$. The quantities a, b, c, d in this expression are of the real or integer type, and the quantities x, y, z are Boolean.

All the Boolean expressions defined so far are termed simple. From the Boolean expressions $\underset{\bullet}{A}$, $\underset{\bullet}{B}$, $\underset{\bullet}{C}$ of which the first expression $\underset{\bullet}{A}$ is simple, we can compose a more complex Boolean expression by use of the service words if, then, and else. The corresponding construction looks like:

if $\mathfrak{C}$ then $\mathfrak{A}$ else $\mathfrak{B}$.

It is assumed that the complex Boolean expression thus defined is $\underset{\bullet}{A}$ if the condition $\underset{\bullet}{C}$ is satisfied (i.e., if the Boolean expression $\underset{\bullet}{C}$ takes the value "true"), and is $\underset{\bullet}{B}$ otherwise. Since of the three expressions $\underset{\bullet}{A}$, $\underset{\bullet}{B}$, $\underset{\bullet}{C}$ only the expression $\underset{\bullet}{A}$ must be simple, the expressions B and C can in turn be composed with the aid of some condition. Finally, in the construction of the simple Boolean expressions, along with the logical values, the variables, relations and functions it is permissible to use any Boolean expressions (both simple and complex) which are enclosed in round brackets.

Thus, recursive constructions of any depth are possible in the determination of the Boolean expressions. For example, the list of Boolean expressions might include the expression $\mathbf{if}\ a > b\ \mathbf{then}\ (\mathbf{if}\ a = b + c\ \mathbf{then}\ B\ \mathbf{else}\ C)\ \mathbf{else}\ \mathbf{if}\ D\ \mathbf{then}\ E\ \mathbf{else}\ F \wedge G(K,L)$, where the lower case letters denote variables of real type, and the capital letters denote variables of the Boolean type, where G(K, L) is a function (Boolean procedure) of the actual parameters K and L. If all this expression is enclosed in round brackets it becomes a simple Boolean expression and as such can be used in further constructions.

The situation is completely analogous in the case of the arithmetic expressions: from the two arithmetic expressions A and B, of which the first is necessarily simple, and the Boolean expression C we can construct a complex arithmetic expression

$$\mathbf{if}\ \mathfrak{C}\ \mathbf{then}\ \mathfrak{A}\ \mathbf{else}\ \mathfrak{B}.$$

The value of this expression is taken equal to A if condition C is satisfied, and equal to B otherwise. Just as in the case of the Boolean expressions, in the construction of the <u>simple</u> arithmetic expressions it is permissible to use not only numbers, variables, and functions, but also <u>any</u> arithmetic expressions (simple or complex) which are enclosed in round brackets. So, for example, the expressions $(\mathbf{if}\ a > b\ \mathbf{then}\ a \uparrow 2\ \mathbf{else}\ a \uparrow 3)$ or $(\mathbf{if}\ a = b\ \mathbf{then}\ a - b\ \mathbf{else}\ a + b) \uparrow (a - b \uparrow 2)$ must be considered simple arithmetic expressions.

All the descriptions presented so far are in essence auxiliary. The basic means for the construction of the algorithms in ALGOL are the so-called operators. ALGOL-60 contains six different types of operators: the assignment operator, the transfer operator, the empty operator, the cycle operator, the procedure operator and the conditional operator. The first five types of operators, in contrast with the last one, the conditional operator, are usually termed unconditional operators.

The assignment operator assigns to particular variables definite values specified by some arithmetic or Boolean expression A. The variables to which the value determined by the expression A us assigned are separated from one another and from this expression by a special separator: =(assignment symbol). All these variables constitute the left part and the expression A constitutes the right part of the assignment operator.

Examples of the assignment operators are: $A: = kl[0]: = v: = n+ +1 + p$; $m: = m + 1$; $B := a>b$; $r[j, zk]:=5 - 3 \times v \uparrow 2$.

In the realization of the assignment operator there must be observed a strictly defined order of performance of the operations. First in order (from left to right) there are calculated the values of the indices (speicficed by the arithmetic expressions, which in this case are termed the subscript expressions) of all the variables of the left part. Then there is computed the value of the arithmetic expression in the right part and the value obtained is assigned to all the variables of the left part (with the already compute subscripts). Thus, for example, the operator: A: = B: = p + q must be performed as z: = p + q; B: = z; A: = z, and not as B: = p + q; A: = p + q. The difference lies in the fact that the value of the arithmetic expression p + q can change with each new calculation (for example, if it contains some function whose values are determined by a procedure which changes in the process of its performance). Therefore in the performance of theassignment the value of the arithmetic expression must be compute only one time, and not with application to each individual assignment.

The operators in ALGOL can be provided with labels, for which use is made of any identifiers or unsigned integers (in the latter case prefixing of zero before the number does not alter the value of the label). However, in order to facilitate the construction of translators (programs for translation from the ALGOL language to machine languages) use of numbers as labels is frequently avoided. The label is separated from the operator by a colon. The operators (labeled or unlabeled) are arranged sequentially one after the other, separated from one another by a semicolon, for example: p: $\underset{\cdot}{A}$; $\underset{\cdot}{B}$; kl: $\underset{\cdot}{C}$ where $\underset{\cdot}{A}$, $\underset{\cdot}{B}$, $\underset{\cdot}{C}$ are operators, and $\underline{p}$ and kl are labels (the operator $\underset{\cdot}{B}$ is an unlabeled operator). The same operator can have not just a single, but as many labels as desired (separated from one another by colons), for example p:A:r7: $\underset{\cdot}{A}$ (here the operator $\underset{\cdot}{A}$ has three labels: $\underline{p}$, A and r7).

Usually the operators in ALGOL are performed sequentially, one after the other, in the order of their writing. Variation in the order of performance of the operators is accomplished by an operator termed the transfer operator. In the simplest form the transfer operator consists of the serivce words go to and some label L. The meaning of the action of this operator consists in that on coming to it a transfer (jump) is made to the operator having L as its label.

In the general case in the transfer operator after the words go to there is placed some designational expression. The label is only one of the simplest examples of the designational expressions. A more complex example of the designational expression is the expression composed of two labels, say L and M, and some Boolean expression

$\mathfrak{C}$: **if** $\mathfrak{C}$ **then** L **else** M . The value of this designational expression is equal to L if the condition C is satisfied, and to M otherwise. In place of the label M (but not in place of the label L) in this expression there can be substituted any complex designational expression, and the similar substitution process can be continued.

As a result there can arise complex recursive constructions for the transfer operator, for example **go to if** $i = 1$ **then** L **else if** $i = 2$ **then** M **else** P . For simplification of this construction use is made of the so-called switch transfer. The switch consists of some identifier and a following so-called index expression enclosed in index (i.e., square) brackets. The index expression is any arithmetic expression which in the computation must every time be rounded off to the nearest integral value.

If, for example, the switch identifier is s and the index expression is the variable (expression) i, then the switch transfer operator s[i] is written go to s[i]. In itself such an expression does not yet have any meaning. In order to give it meaning it is necessary, in addition to the expression s[i], which we shall term the switch indicator and consider as a simple designation expression, to also introduce the so-called switch description, usually placed together with the description of the types of variables, arrays and procedures (functions). The switch description begins with the service word switch, after which goes the switch identifier, then the assignment symbol : = and, finally, the so-called switch list, i.e., the list of designational expressions separated from one another by commas. For example: **switch** $s := L,M,P$ (where L, M, P are labels).

On encountering the switch transfer operator, for example **go to** $s[i]$, we compute the corresponding index expression, substituting in it the current values of the variables (say, i = 2). After this we turn to the description of the switch with the same identifier s and accomplish the transfer with respect to that designational expression in this description whose sequential number in the list coincides with the found value of the index expression. In the case considered there will be performed a transfer with respect to the label M, i.e., with respect to the *second* element of the switch list.

If the value of the index expression in the switch indicator cannot be calculated (as a result of the fact that values have not yet been assigned to certain variables) or if this calculation leads to a number which is not a number of any element of the switch list, then the transfer operator is not performed and there is immediately performed the operator following it. In the example considered above the values of the index expressions equal to 4.0 or −1 do not lead to the objective. However, the values of the index expressions equal to 2.2 or 2.7 lead (after their roundoff) to transfers with respect to the second or, correspondingly, with respect to the third element of the switch list (i.e., with respect to the labels M or P).

A label, switch indicator or any designational expression enclosed in round brackets is a *simple designational expression*. From the two designational expressions $\underset{\cdot}{A}$ and $\underset{\cdot}{B}$ (of which the first is necessarily simple) and the Boolean expression $\underset{\cdot}{C}$ we can compose the complex designational expression **if** $\mathfrak{C}$ **then** $\mathfrak{A}$ **else** $\mathfrak{B}$, which coincides with the expression $\underset{\cdot}{A}$ in the case of satisfaction of the condition $\underset{\cdot}{C}$ and with the expression $\underset{\cdot}{B}$ otherwise. Thus, for the designational expressions exactly the same recursive constructions are found to be possible as for the arithmetic (or Boolean) expressions.

The third type of operator used in ALGOL is the so-called *empty operator*, which does not perform any operation and designated an empty set of symbols. Usually the empty operator is provided with a label and serves for the return using this label (as a result of the application of the transfer operator) to the required segment of the program. The use of the empty operator with a label is absolutely necessary, for example, in the case when it is necessary to perform a transition from the middle of a program to its end (not to the following nonempty program operator but precisely to the end of the program). Just as in the other operators, there must be a colon placed after the label in the empty operator.

Of very great value in the construction of programs in the ALGOL language are the so-called cycle operators, whose meaning is that some operator (or group of operators) is performed some number of times in sequence. The cycle operator consists of the cycle heading and the operator itself (which can be any operator), which is performed multiply in the cycling process.

The cycle heading begins with the service word for and terminates with the service word do. After the word for there stands the identifier of that variable which changes in the process of the performance of the cycle. This variable is termed the cycle paramter. Following it, after the assignment symbol : ≐, there is the so-called cycle list, the listing of those values which the variable must take during the cycle operating time. The cycle list consists of one or several elements of the cycle list, separated from one another by commas. In the simplest case the arithmetic expressions (in particular, simply numbers) are used as the elements of the cycle list. For example, the cycle operator **for** i: = 1,2,3 **do** $a[i]$: = $i \uparrow 2$ performs the sequential assignments $a[1]$: = 1; $a[2]$: = 4; $a[3]$: ≐ 9 . The length of the cycle in this case is equal to 3.

If the cycle parameter must take not three, but, say, a thousand different values, then the listing of all these values in the cycle list would be excessively cumbersome. In this case we use as the cycle elements special constructions, each of which gives immediately some set of values of the cycle parameter.

In ALGOL use is made of two types of such construction. The first type is constructed with the use of the service words step and until and has the form A step B until C, where A, B and C are arithmetic expressions. An element of the cycle list of this type gives the values of the cycle parameter as follows: in the first step the cycle parameter is assigned the value of the arithmetic expression A, in the second step – the value of the arithmetic expression $A_1 = A + B$, in the third – the value $A_2 = A_1 + B$ etc., until the next value $A_n = A_{n-1} + B$ exceeds the value of the arithmetic expression C.* This value is not assigned to the cycle parameter and the cycle for it is not performed. The cycle list is considered to be exhausted and, consequently, there must be performed a transfer to the operator directly following the cycle operator.

As an example of the construction described let us consider the cycle operator having the form **for** $i:=12,4$ **step** -1 **until** $0,-5$ **do** $a[i]:=i+10$. This operator performs the sequential assignment: a[12]: = 22; a[4]: = 14; a[3]: = 13; a[2]: = 12; a[1]: = 11; a[0]: = 10; a[-5]: = 5. We note that in the first example the cycle list element 4 **step** -1 **until** 0 describes an arithmetic progression (with a difference equal to minus 1), however in the general case the step represented by the arithmetic expression B (standing after the word step) can be a variable, varying with every new repetition of the cycle.

The second type of cycle list element is given with the aid of the arithmetic expression A, the Boolean expression B and the service word while, written in the sequence: A while B. This element provides the sequential assignment to the cycle parameter $\underline{t}$ of the values taken by the arithmetic expression A until the condition B is satisfied (i.e., until the expression $\underset{\cdot}{B}$ has the value "true"). If with a succeeding performance of the cycle the condition $\underset{\cdot}{B}$ ceases to be satisfied, then the cycle operator is not performed and there is accomplished a transfer to the operator directly following it. We note that in the construction described it is forbidden to use the word step, so that the expression of the type $\mathfrak{A}$ **step** $\mathfrak{B}$ **while** $\mathfrak{C}$ is not encountered in ALGOL.

With the method described above for the construction of the cycle operator, we can accomplish the repetition of only one single operator which follows immediately after the word do. If it is required to repeat in a particular cycle not one single operator, but some sequence of operators A_1; A_2; :..; A_k, then this sequence is enclosed in special <u>operator brackets</u>, considering it after this as a single <u>complex operator</u>.

As the operator brackets we make use of the pair of service words begin and end, so that the complex operator is written **begin** A_1; A_2; ...; A_k **end** . The complex operators, just as the conventional, can be provided with labels (one or several).

Along with the complex operators, in ALGOL use is made of the so-called <u>blocks</u>, differing from the complex operators in that ahead of the operators appearing in the block, directly after the word begin, there is placed a description of the types of certain quantities (identifiers) which are encountered in this block. In this case the quantities described in the block are localized only in the given block and, generally speaking, they lost their value (become indeterminate) with departure from the block. If we wish to retain the value of certain of the quantities described in the block after departure from the block for purpose of using them on repeated reference to the block, then to the description of their types there is added the word own. An example of the block: **begin own real** x; **integer** n; $n := s+i$; $x := a \uparrow n$ **end** .

We note that both the complex operators and the blocks can include in themselves other blocks and complex operators, permitting any recursive depth of such constructions. The identifiers used within the block for the designation of the improper quantities can be used outside the block for the designation of any other quantities which are not accessible for this block (i.e., which do not figure in the given block and are not subjected to any transformation in it). Identifiers which are not described in a block cannot be localized in it and, consequently, represent the same objects both inside the block and outside of it.

The labels are always assumed to be localized within the block in which they are encountered, so that entry into the block can be accomplished only through its origin. No transfer oprrator located outside the block can accomplish transfer to any operator within this block.

In exactly the same way it is not possible to accomplish a transfer with respect to a label located within a cycle operator with the aid of a transfer operator acting from outside the cycle. We note also that with exit from the cycle operator as a result of exhaustion of the cycle list the value of the cycle parameter is considered indeterminate. If, however, the exit from the cycle is accomplished as a result of the transfer operator contained in the composition of the operator (or block) which is repeated in the given cycle (i.e., standing after the word do) then the value of the cycle parameter is retained just as it was immediately before the performance of the transfer operator.

All the simple operators described above and also the procedure operator described below, and all the complex operators and blocks to system of so-called unconditional operators. In ALGOL there are introduced two other types of conditional operators, using the condition **if** $\mathfrak{B}$ **then** (where B is a Boolean expression) which is analogous to the condition used in the construction of the complex arithmetic, Boolean and designational expressions.

The operator "if" is constructed from the described condition and the following unconditional operator which is performed in the case when the condition is satisfied, and is bypassed (not performed) otherwise, Example: **if** $a > v$ **then begin** $A := n$; **go to** L **end** . The complex operator **begin** $A := n$ **go to** L **end** is performed if and only if the condition $a > b$ is satisfied.

The conditional operator proper is obtained by the addition to the operator "if" the service word else and the following arbitrary operator (possibly also conditional). This operator must be performed in the case when the condition in the "if" operator is not satisfied. The general structure of the conditional operator thus has the form

if $\mathfrak{B}$ **then** A_1 **else** A_2,

where B is any Boolean expression, A_1 is an unconditional operator, A_2 is any operator.

The so-called procedure operators are of essential importance in the construction of ALGOL. Procedure is the term given to some ensemble of operators designated by some identifier, termed the procedure identifier. In ALGOL the procedures play the same role as the subroutines in conventional programming, permitting the acceleration of compilation of complex programs by means of the use of precompiled standard programs. Decoding of the procedure (actual writing of the operators composing it) can be performed either in the ALGOL language or directly in the language of the corresponding universal digital machine.

The procedure operator (if we are not considering the familiar standard procedures) must be described in advance. This description is accomplished with the aid of the service word procedure, after which there follows the so-called procedure heading, i.e , the procedure identifier, and after it (in round brackets) a list of the so-called formal parameters of the procedure, i.e., the identifiers separated from one another by special limiters, specifically commas, or by limiters of the form) letter line: (. For example, procedure sin (x) or procedure A (x, y) pressure: (p). The first procedure has one formal parameter (x), and the second has three formal parameters x, y, p. Procedures without parameters are also possible. Their heading consist only or procedure identifiers, not accompanied by following brackets. The procedure itself (the so-called body of the procedure) is written out after the procedure heading in the form of some operator.

The procedure operator itself, or, more exactly, the procedure derivation operator, is written in the same form as in the procedure description, but now without the word procedure ahead of it and under the condition that the formal parameters of the procedure are replaced by its so-called actual parameters. The brackets and limiters are the same as in the sescription of the corresponding procedure. The performance of the procedure operator consists in the assignment of all the formal parameters of the values of the corresponding actual parameters, or replacement of the formal parameters by actual and subsequent performance of the procedure.

As the actual parameters use can be made of any expressions (arithmetic, Boolean or designational), array identifiers and switch identifiers, identifiers of any procedures and, finally, the so-called lines.

The lines are any sequences of symbols enclosed in special "line" brackets ‘ ’ . These brackets can also be used within a line. Example: $‘10 + - [[x \uparrow \uparrow b :‘ \wedge \vee \supset ’ \equiv Ad’$. The lines can be used as the actual parameters only of those procedures which are written in machine codes, and not in the ALGOL language. Most frequently their use is limited to the special procedure punch (x) which performs the printing or perforating of the actual parameters, which are represented in place of the formal parameter <u>x</u>. If, in particular, in place <u>x</u> there is substituted some line, then the procedure punch performs the extraction of all the symbols of this line from the machine for printing or perforating. Therefore the line can contain not only the "analog" but any other symbols which the considered printing or perforating device is capable of realizing.

In the procedure description there is also indicated the type of its formal parameters. With the substitution of the actual parameters their types must coincide with the types of the corresponding formal parameters. In order to avoid ambiguity in such a substitution, it is usually necessary to perform a replacement of those identifiers localized within the procedure which coincide with the identifiers occurring in the actual parameters being substituted.

We note that among the procedure parameters there appear, generally speaking, both the input and output (obtained as a result of the performance of the procedure) quantitites of this procedure. If as a result of the performance of the procedure there is obtained only one quantity (number or logical value) then it is natural to denote this quantity by the identifier of its procedure (together with the line of actual parameters). In this case the corresponding procedure is termed a function (see above) and in its description there is placed ahead of the word procedure the word designating the type of output quantity of this procedure. Example: real procedure sin (x). It happens frequently that in the procedure some formal parameter $\underline{x}$ participates in several transformations. For example, the parameter $\underline{x}$ in the procedure sin(x) with the calculation of the sine using a series is raised sequentially to the powers 3, 5, 7 etc. If in the substitution this parameter is replaced by a quite complex expression (actual parameter), say, $x: = (a + b) \times (a - b)$, then in the development of the procedure we can encounter the necessity for the repeated computation of this expression in every case when a particular operation is performed with the parameter $\underline{x}$. Naturally it is simpler to compute the value of $\underline{x}$ ahead of time (prior to entry into the procedure) and substitute this value in place of it.

In the automatic translation of a program from the ALGOL language to machine language, it is necessary every time to communicate to the translator (programming program) which parameter values must of necessity be computed prior to substitution into the procedure. All such parameters in the description are labeled with the special service word value and are placed after the ensemble of formal parameters of the procedure heading before the description of their types (the so-called specifications). For example, in place of the description real $\underline{x}$; integer $\underline{n}$ there may appear the description **value** *n*; **real** *x*; **integer** *n*.

We shall usually supplement ALGOL with two procedures which are not defined in the descriptions for the entry and output of information from the machine. The first procedure is always assigned the same identifier read, and the second is assigned the identifier punch; the actual parameters of each of these procedures will be considered either some quantities of the type real, integer or Boolean, or the array identifier of any of these types.

We shall make some other remarks. Normally every program is ALGOL is represented in the form of a block, i.e., is enclosed in the statement brackets begin – end. To facilitate the reading of the "analog" programs there can be introduced into them the so-called commentaries, i.e., clarifications for the programmer, which have no intrinsic meaning in the ALGOL language and which are therefore not accepted by the translator in automatic programming.

The commentary is considered to be every sequence of symbols (not necessarily "analog") beginning with the service word comment after a semicolon or the word begin, terminating with a semicolon and not containing within itself other occurrences of a semicolon. Any sequence of symbols following after the word end to a semicolon or to the end of the program is also considered a commentary if it does not contain the words end and else, or a semicolon. For example, in the expressions comment text; begin comment text; end text; the word "text" is a commentary. From the point of view of the "analog" programs the first expression is equivalent to an empty place, the second – to the word begin, and the third – to the word end.

For the electronic digital machines with small and medium capacity the ALGOL-60 language is excessively complex to permit organizing effective translation from it to the machine language. Therefore there has been proposed the simplified variant of ALGOL which has been termed SMOLGOL-61.*

The simplification amounts to the following. First, the alphabet is limited to either only lower-case or only capital letters of the Latin alphabet. Second, we exclude from consideration the logical operations of implication and equivalence, and also the service words while, Boolean, true and false. Thus, the use of the logical values "true" and "false" is not permitted. The use of the identifiers for the designation of the logical quantities is also prohibited. The Boolean variables can be introduced by the programmer indirectly, with the aid of the replacement of the logical values "true" and "false" by the whole numbers 1 and 0. The use of Boolean expressions is permitted only in conditions. If in ALGOL the value of some Boolean expression B was assigned some identifier P — P : = B, then in SMOLGOL there must correspond to it the assignment of the form P: = **if** B **then** 1 **else** 0, where in the right side there now stands an arithmetic expression - rather than a Boolean expression.

Further, the length of the identifiers is limeted to five letters. More exactly, identifiers in which the first five letters coincide are considered identical in SMOLGOL. In the arithmetic expression $a \uparrow b$ negative values for the exponent b are not permitted in the integer type quantities a and b. Whole numbers are not used as labels. The step in the cycle operator must either remain positive at all times, and in the latter case the symbol "minus" must be placed explicitly ahead of the expression which specifies the step. In cycle list there must be only one step-until element.

In all the procedures, except the input and output procedures, use cannot be made of lines as actual parameters. No procedure can be called on before it has been described. The possibility of using one procedure within another is excluded if they were described in the same block. A second callup of the same procedure is forbidden until its first call has been completely terminated. For example, use cannot be made of recursive calls of the procedure F(u, v) of the form F(x, F(x, y)). But repeated use of a procedure after its termination is not prohibited, so that the expression ln(ln x) is completely acceptable in SMOLGOL. If the procedure P is an actual parameter of another procedure, then all the parameters of the procedure P must be described as value.

Standard procedures for the finding of the sign and absolute value of a number cannot be used as actual parameters in any procedures. If it is necessary to use them in this fashion, then they must first be described as functions, i.e., write out the expression **real procedure** *abs*(*x*); **value** *x*; **real** *x*; **begin** *abs*: = *abs*(*x*) **end** (and similarly for integer procedure sign (x)). Neither procedures nor their formal parameters can be of the Boolean type.

The variables themselves cannot relate to portions of the program outside of the block in which they are defined. The boundaries of arrays must be constant. The elements of the switch lists in the switch descriptions can be only labels and not any designational expressions.

The descriptions of the procedures which are called in any block must be accomplished after the description of the types, switches and arrays of the corresponding block. The procedure identifiers can appear within a procedure only in the case when they are the left parts of the corresponding assignment operators. Some other limitations also exist.

We note that any program written in SMOLGOL can also be considered as an "algol" program. Generally speaking, the reverse is not true.

§5. EXAMPLES OF PROGRAMMING USING ALGOL-60

Let us consider first a very simple example which has already been used in §3 of the present chapter as an illustration of the principles of programming in machine languages. This is the calculation of the value of the sum $\sum_{k=1}^{m}\frac{1}{k^2}$.. The corresponding "algol" program can be written in the form

```
begin real s; integer m, k;
    read (m); s: = 0;
    for k: = 1 step 1 until m do
    s: = s + 1/k↑2;
    punch (s)
end.
```

The procedures read(m) and punch(s) respectively provide for the entry of the required information (upper limit of the summation) and the output of the result, i.e., the value of the desired sum. It is easy to see that the program written in ALGOL is far more visible and understandable than the machine program written in §3 which solves the same problem.

Programs for the other examples considered in §3 can also be written quite lucidly and clearly. The computation of the scalar product of two (real) vectors (a1, a2,..., an) and (b1, b2, ..., bn) is represented in the form

```
begin integer n; read (n);
begin real s; integer i; real array a[1 : n], b [1 : n];
    read (a); read (b); s: = 0:
    for i: = 1 step 1 until n do
      s: = s + a [i]×b [i]:
      punch (s)
 end end.
```

Multiplication of the vector (b1, b2, ..., bn) by the matrix $\|A_{ik}\|$ can be represented in ALGOL by the program

```
begin integer n; read (n);
begin integer i, k; real array s[1 : n], b[1 : n], A[1 : n, 1 : n];
    read (b); read (A);
    for k: = 1 step 1 until n do
    begin s [k]: = 0;
    for i: = 1 step 1 until n do
    s[k]: = s [k] + b[i] × A [i, k];
    end
    punch (s);
end end.
```

In this program one cycle operator occurs in another. Internal operator brackets are introduced, since in the first (outer) cycle it is necessary to perform a sequence consisting of two operators.

The two phrases written at the end of the program are a commentary which does not actually enter into the program and is not accepted by the translator (programming program).

We note that the ALGOL language is a universal algorithmic language and is therefore suitable for the writing of any algorithms. In addition, as was noted above, all the programs written in ALGOL can be realized (under the condition of the use of a sufficiently large memory volume) by any universal electronic digital machine.

We shall make use of the last circumstance to illustrate that the universal elecrtonic digital machines can perform not only the conventional algorithms, but also algorithms with random transfers and <u>any self-organizing systems</u> of algorithms.

In order to have the possibility of constructing in ALGOL any desired random algorithms it is sufficient to introduce into it a special procedure which we shall designate as <u>random (a, b)</u>. With each referral to this procedure it generates some random number belonging to the segment [a, b]. Here it is assumed that the selection is made on the basis of a uniform distribution law according to which all the numbers of the indicated segment are considered equally probable. The random numbers themselves are assumed to be of the integer or real type depending on what type is assigned to the formal parameters (segment bounds) <u>a</u>, <u>b</u>. Of course both these parameters must be of the same type.

The method of construction of the procedure itself can vary over quite wide limits. We can, for example, simply write into the machine memory a table of random numbers and construct the procedure for their sequential selection. In many cases a special random number unit is appended to the electronic digital machine. In this case the procedure consists in the selection of the numbers generated by the indicated unit and their subsequent transformation for the purpose of reduction to the given interval [a, b].

Wide use is also made of the various procedures which generate sequences of the so-called pseudorandom numbers. For the formation of such a sequence we can make use, for example, of the following technique: some positive number a_1 is selected and squared. In resulting number $a_2 = a_1^2$ there is selected some group of digits (usually not the highest or lowest). The number b_2 formed by these digits is taken as the first pseudorandom number. Squaring the number b_2, we obtain the new number $a_3 = b_2^2$ which we treat just as we did the number a_2. Continuing this process we obtain the required sequence of pseudorandom numbers.

The sequence constructed in this fashion, if its length is not too great, can be considered practically random. However, with a long sequence there occur various sorts of cyclings (cyclic repetitions of previously encountered pieces of the sequence) which is what differentiates the pseudorandom sequences from the purely random. However, for each concrete case there can be selected that procedure for the generation of the pseudorandom sequence which form the purely random sequences.

With the aid of the indicated procedures the problem is completely resolved of the realization on the universal electronic digital machines of any random algorithms. The problem of the realization of the self-organizing systems of algorithms on the machines is actually even simpler, since in this case, as a rule, we do not have to resort to any special procedures. Such a realization is accomplished by the usual methods, with the aid of programs written in the ALGOL language. We shall present examples of the sort of self-organizing systems described in the preceding chapter.

As the first example let us consider the self-adaptive control algorithm based on the method of steepest descent. The task of this algorithm is the generation of those arrays A[1: n] of numbers (control actions) such that the criterion f will have the smallest possible value. The criterion f is a known function of certain parameters (indications) of instruments which monitor the process whose values in the form of the corresponding array B[1: m] are periodically introduced into the algorithm.

The parameters composing the array B (control results) vary as a result of the variation of the controlling actions (array A) and also as a result of other factors relating to the controlled process and which do not depend on the control algorithm. The factors in question here reduce to the variation of certain uncontrolled parameters, where the nature of this variation is not known ahead of time to the controlling algorithm. It is easy to see that the described control algorithm accomplishes extremal regulation (with respect to the criterion f) whose quality will be better the smaller the ratio of the time for the algorithm to determine the optimal controlling actions (array A) to the average time in the course of which there occurs a sensible variation of the uncontrollable parameters. It is not difficult to verify that the control algorithm in question can be described in the ALGOL language by the following program:

```
begin integer i, j, k; real y, s;
    B[1:m], P[1:n];
    L: read (B); y: = f(B); s: = 0;
    for i: = 1 step 1 until n do
```

```
          begin A[i]: = A[i] + d;
          punch (A); read (B);
          P[i]: = (f (B) − y)/d;
          s: = s + P[i]↑2;
          A[i]: = A[i] − d
          end;
          r: = d/sqrt (s);
          for j:= 1 step 1 until n do
          A[i]: = r × P[i] + A[i];
          punch (A); read (B);
          if aBs (y − f(B)) > l then go to L else
          for k:= 1 step 1 until n do
          A[i]: = A[i] − r × P[i];
          punch (A);
          go to L
       end.
```

In the construction of the program the quantity d is the steepest descent step and the quantity ℓ is the accuracy of achieving the minimal value of the criterion f. The function sqrt(s) is equal to the square root of s taken with a plus sign. The array P[1: n] gives the relative magnitudes of the optimal increments of the values of the control actions A[1: n] at each step of the steepest descent process. It is assumed that the quantities d, ℓ, the real procedures f(B) and sqrt(s) and the initial values of the components of the array A[1: n] were introduced into the algorithm previously (prior to the instruction with the label L).

Now let us consider the algorithm with performs the operation of a discrete α-perceptron P. Let us assume that the perceptron P has a retina consisting of N receptors and is designed for the recognition of two patterns. The A-elements are (1, 1, 1)-neurons, the reward constant is equal to unity, and the penalty constant is equal to zero. The image projected onto the retina is the Boolean array r[1: N], which is read externally with the showing to the perceptron of each new image. Also sensed extrenally is the Boolean quantity a which is the applied signal on the correctness (truth) or incorrectness (falsity) of the response p given by the perceptron. The response p is simply the number of the pattern to which the perceptron assigns each image shown to it. We use sf and ss to designate the output signals of the summators of the first and second patterns.

Let us assume further that the number of neurons of both the first and second image is equal to n. We use xf[i] and yf[i] to denote the numbers of the retina receptors to which there are connected respectively the exciting and inhibiting inputs of the the ith neuron of the first pattern, we use xs[i] and ys[i] to denote the corresponding numbers of the receptors for the ith neuron of the second pattern, and vf[i] and vs[i] to denote the weights of the ith neurons of the first and second patterns (i = 1, 2, ..., n). With these assumptions, the algorithm which performs the work of the perceptron P (in the learning regime) can be written in the form

```
begin integer p, i, j, k; real sf, ss; Boolean a;
    Boolean array r[1:N];
    L: read (r); sf: = 0; ss: = 0;
    for i: = 1 step 1 until n do
    begin if r[xf[i]] ∧ ¬r[yf[i]] then
    sf: = sf + vf[i];
    if r[xs[i]] ∧ ¬r[ys[i]] then
    ss: = ss + vs[i]
    end;
    if sf > ss then p: = 1;
    if sf < ss then p: = 2;
    if sf = ss then go to L;
    punch (p); read (a);
    if a ∧ (p = 1) then for j: = 1 step 1 until n do
    if r[xf[j]] ∧ ¬r[yf[j]] then vf[j]: = vf[j] + 1;
    if a ∧ (p = 2) then for k: = 1 step 1 until n do
    if r[xs[k]] ∧ ¬r[ys[k]] then vs[k]: = vs[k] + 1;
    go to L
end,
```

It is assumed that the arrays $xf[1:n]$, $yf[1:n]$, $xs[1:n]$, $ys[1:n]$, $vf[1:n]$ and $vs[1:n]$

are introduced into the program ahead of time and that the last two arrays (weights of the neurons of the first and second patterns) do nto consists of only zeros. Otherwise the pattern summators would generute continuously signals (sf and ss) which are equal to zero and, since it is assumed in the program that with equal signals of the summators the perceptron will not generate any signal p, the learning process (and, in general, any variation of the weights) would not exist.

This last limitation can be avoided if the teacher does not simply supply a reward signal but communicates to the perceptron the true number of the pattern to which the image being shown to the perceptron belongs. This is precisely the method of functioning of the perceptron in the learning regime which was considered in the preceding chapter. Let us indicate the changes in the program described above which must be made with application to the new type of signal a.

In the description the quantity a must be declared as a quantity of the integer type and not Boolean. The program changes can be reduced to the following. After the operator **if** $sf < ss$ **then** $p := 2$ in place of the operator **if** $xf = ss$ **then go to** L it is necessary to use the operator **if** $sf \neq ss$ **then** *punch*(p) . Then in the conditional operators following after the operator read(a), the conditions **if** $a \wedge (p = 1)$, **if** $a \wedge (p = 2)$ must be replaced by the conditions if a = 1 and if a = 2 respectively.

It is also not difficult to describe the changes in the original perceptron program which must be made in order to simulate the perceptron self-learning regime rather than the learning regime. To do this the quantity a is completely excluded from the program together with the corresponding operator read(a). In the following conditional operators there must be added to the conditions standing after the service words do the terms $\wedge (sf > ss)$ and $\wedge (sf < ss)$ respectively.

Thus, in both the learning and self-learning regimes the α-perceptrons can be easily simulated in the ALGOL language and consequently can be simulated on the universal electronic digital machines. It is easy to see that the same holds for any modifications and generalizations of the perceptron circuit.

Let us now describe using the ALGOL language still another self-organizing algorithmic system which simulates the process of biological evolution and the formation of new species. The modeling of such a system (although somewhat different) on the universal electronic digital machine has been accomplished by Letichevskiy [49].

Let us consider a discrete space consisting of a finite set of points with the numbers from 1 to $\underline{n}$ inclusive. Let us assume for simplicity that this set is cyclically ordered. In other words, for each point $\underline{i}$ we define the two points neighboring with it – the point directly preceding it bi and the point directly following it fi. If $i \neq 1$, then $bi = i - 1$; for $i = 1$ we set $bi = n$. Similarly, if $i \neq n$, then $fi = i + 1$, and for $i = n$ we set $fi = 1$.

To every point $\underline{i}$ of the space we assign some state s[i] which can take any integral value from 0 to $\underline{k}$ inclusive. If $s[i] = 0$ the corresponding point is considered "lifeless." If, however, $s[i] \neq 0$, then we assume that at the point $\underline{i}$ there is some "living being" in the state s[i]. As the such "living beings" in the considered model we select abstract automata with the same number of states (equal to $\underline{k}$) but, generally speaking, with different transfer and output tables.

In addition, for each point $\underline{i}$ of our space there is given the number F[i], equal to 1 or 0 in accordance with whether or not ther is "food" at the $\underline{ith}$ point. In the case of the existence of "food" at a particular point its supply is assumed so large (or self-replenishing) that the automaton located at this same point will practically not alter the supply in the course of its "feeding." The array F is altered at each successive cycle of operation of the algorithm in accordance with some "law of nature" which we shall assume to be located outside of our algorithm.

In addition to the state s[i] itself of the automaton occupying the point i, with this automaton we associate also two other numbers, namely the indications of its "life" counter L[i] and the so-called "hunger" counter H[i]. The quantity L[i] increases by unity at each cycle of operation of the algorithm, and after its value exceeds some prespecified threshold ℓ, the corresponding automaton transfers into the zero state, i.e., simply speaking, it is destroyed (simulating thereby natural death).

The quantity H[i] increases by unity if F[i] = 0 (i.e., in the case when the automaton is located at a point of the space without "food") and decreases by unity in the opposite case, without, however, taking negative values (in the case when H[i] = 0 we set H[i] – 1 = 0. When the quantity H[i] exceeds some level h which is fixed in advance, the corresponding automaton transitions into the zero state (thereby simulating death from hunger).

The input signals of the automaton located at the point i are the states s[bi] and s[fi] of the neighboring points, and also the signals F[bi], F[i], F[fi] on the presence or absence of food, both at the point i itself and at the neighboring points. The output signal m is the so-called motion of the automaton, i.e., in other words, the increment of the number of the spatial point occupied by the automaton in the given automaton operating cycle. We shall consider that the quantity m can take only three different values: 0, 1 and –1.

In view of the presence of five input channels, the switching and output tables of each automaton can be specified in the form of sixdimensional arrays. For the specification of th switching and output tables of all the automata at the same time we make use of the seven-dimensional arrays SP[1: n, 1:k; 0:k, 0:k, 0:1, 0:1, 0:1] and M[1:n, 1:k, 0:k, 0:k, 0:1, 0:1, 0:1] respectively. The first index in each of these arrays indicates the number of the cell _i_ occupied by the automaton, the second indicates the state s[i] of this automaton, the third and fourth indicate the states of the neighboring points s[bi] and s[fi], the fifth, sixth and seventh indices are the signals F[i], F[bi] and F[fi] on the presence or absence of "food" at the point itself and at the neighboring points. For the motion, given by the output table M, we shall not introduce any limitations in the table itself, however the performance of the corresponding motion will be accomplished only in the case when the point to which the automaton is shifted is not occupied by any other automaton.

The variation of the indications of the "life" and "hunger" counters is accomplished after the performance of the motion and the transfer of the automaton into the new state. If in this case there does not occur "death" of the automaton, and its motion is nontrivial (i.e., the automaton does not remain at the previous location), then with fulfillment of certain additional conditions there takes place "reproduction" of the automaton by means of fission. In this case the shifted automaton A completely retains its structure with the exception of the fact that on its "life" counter there is established a value equal to zero. And at the place occupied by the automaton A prior to this there appears its "double," differing from A only in that in each of the two arrays which specify the transitions and outputs of the automaton A, one number (respectively the new state or the motion of the automaton) is replaced by a random number. The "life" counter of the new automaton is also set to zero.

Additional conditions for the possibility of reproduction, which we discussed above, are that the indications of the "life" counter be included between two a priori fixed numbers $\ell\ell$ and ℓu, while the indication of the "hunger" counter does not exceed some number hu, also fixed ahead of time.

For the formation of the random numbers we fix the special integral procedure *random(a, b)*, which delivers at each call some integral number located on the closed segment [a, b]. In this case all the whole numbers of the indicated segment are considered equally probable. The methods of construction of such procedures were described above.

The algorithm which we have described in one operating cycle must perform a scan of all the points of our space, performing at these points the changes listed above. After finishing each cycle the algorithm must read through all the new values of all the components of the array F[1:n] and begin the performance of the following cycle. We shall accomplish the count of the number of cycles with the aid of the special quantity t. When this quantity reaches the value p which is fixed in advance the algorithm must terminate its operation.

To facilitate the programming of the described algorithm in the ALGOL language, we introduce three blocks which describe the process of the movement of the automaton located at the ith point, the process of its "death" and the process of its "reproduction." For brevity let us denote these blocks by B_1, B_2 and B_3 respectively, and we write for each of them the corresponding program in ALGOL.

The block B_1:

```
begin integer m, j, bs, fs, f, bf, ff;
            m: = M[i, s[i], s[bi], s[fi], F[i], F[bi]F[fi]];
```

```
        pi: = if $i+m>n$ then 1 else $i+m$;
        if $s[pi]\neq 0$ then pi: = i comment pi is the number of
the point to which the considered automaton is displaced;
        L[pi]: = L[i] + 1;
        H[pi]: = if F[pi] = 0 then H[i] + 1 else if $H[i]\neq 0$
        then H[i] − 1 else 0;
        if $pi\neq i$ then
        for j: = 1 step 1 until k do
        for bs: = 0 step 1 until k do
        for fs: = 0 step 1 until k do
        for f: = 0,1 do for bf = 0,1 do for ff: = 0.1 do
        begin SP[pi, j, bs, fs, f, bf, ff]: = SP[i, j, bs, fs, f, bf, ff];
        M[pi, j, bs, fs, f, bf, ff]: = M[i, j, bs, fs, f, bf, ff]
        end;
        s[pi]:=SP[i, s[i], s[bi], s[fi], F[i], F[bi], F[fi]]
end.
```

The block B_1 accomplishes the displacement of the automaton from the point i to the point pi, its translation into the new state (defined by the situation at the moment the automaton is located at the point i), the change of the indications of the "life" and "hunger" counters and the rewriting of the arrays which specify the switching and output functions of the automata, with the objective of bringing them into correspondence with the new location of the considered automaton. The values of i and pi are retained with departure from the block.

The block B_2 is very simple: **begin** s[pi]: = 0 **end**.

We note that the program will be constructed so that the values of L[pi] and H[pi] at the point pi which are retained after death of the automaton, and the values of the corresponding components of the arrays SP and M cannot lead to errors in the furture. This is achieved as the result of the fact that with repetition of the program the listed quantities, before being used, are defined anew, since they of necessity will occur first in the left parts of the corresponding assignment operators.

The "reproduction" block B_3:

```
begin integer rj, rbs, rfs, rf, rbf, rff, j, bs, fs, f, bf, ff;
        Boolean B;
        rj: = random (1, k);
        rbs: = random (0, k);
        rfs: = random (0, k);
        rf: = random (0, 1);
        rbf: = random (0, 1);
        rff: = random (0,1);
        for j: = 1 step 1 until k do
        for bs: = 0 step 1 until k do
        for fs: = 0 step 1 until k do
        for f: = 0,1 do for bf: = 0,1 do for ff: = 0,1 do
    begin B: = j = rj ∧ bs = rbs ∧ fs = rfs ∧ f = rf ∧ bf = rbf ∧ ff = rff;
        SP[i, j, bs, fs, f, bf, ff]: = if B then random (1, k)
        else SP[pi, j, bs, fs, f, bf, ff];
        M[i, j, bs, fs, f, bf, ff]: = if B' then random (— 1, 1)
        else M[pi, j, bs, fs, f, bf, ff]
    end
    end.
```

The entire program for the modeling of the evolution process is now represented as follows:

```
begin integer t, i, bi, fi, p, q, pi;
        integer array F[1:n], S[1:n], L[1:n], H[1:n],
        SP[1:n, 1:k, 0:k, 0:k, 0:1, 0:1, 0:1], M[1:n, 1:k, 0:k,
        0:k, 0:1, 0:1, 0:1];
        for q = 1 step 1 until n do
        L[q]: = H[q]: = 0;
        t: = 0; i: = 1; read (S); read (SP); read (M);
        Q: read (F); fi: = if i ≠ n then i + 1 else 1;
        if S[i] = 0 then begin i: = fi; go to P end;
        bi: = if i ≠ 1 then i — 1 else n;
                          | block 𝔅₁ |;
        if L[pi] > l ∨ H[pi] > h then
                          | block 𝔅₂ |;
```

```
                if $pi \neq i \wedge H[pi] \leqslant hu \wedge L[pi] < lu \wedge L[pi] > ll$ then
                                    | block $\mathfrak{B}_3$ |;
              if $pi \neq fi$ then $i := fi$ else $i :=$ if $fi \neq n$ then $fi + 1$ else $1$;
                P: if $i = 1 \vee pi = 1$ then $t := t + 1$; if $t \neq p$ then go to Q
        end.
```

We note that the program which we have constructed is not economical from the point of view of the use of the memory and the necessity for rewriting of the multi-dimensional arrays. We can achieve a far more economical program construction if we introduce numeration of the auromata and use in the arrays L, H, SP and M the number of the corresponding automaton in place of the number of the spatial point.

In the real modeling of the evolutionary process on a universal electronic digital machine in [43], use was made of a program with more limited capabilities, nevertheless, the experiments conducted showed that even with these conditions the quality of the simulation was quite satisfactory. For relatively simple "laws of nature" the process of adaptation of the automata to the surrounding medium and the formation of stable "species" were observed after several tens of thousand of cycles of operation of the algorithm and the replacement of the corresponding number of "generations." Initially the transition and output tables of the automata (arrays M and SP in our case) were specified arbitrarily. In the evolution process there took place a "dying out" of the poorly arranged automata and the appearance of forms which were better adapted for "life" under the given conditions.

Manuscript Page No.	[Footnotes]
361	The diode matrices are two systems of conductors, usually termed buses, a part of which is interconnected by diodes, i.e., elements which pass current in only one direction.

384 An attempt is usually made to avoid such roundoffs in ALGOL, since natural roundoff of the quantity 23.5 in some machines leads to 23, and in others to 24.

396 If the step B is negative, then the value of the expression $C - A_n$ is taken with reversed sign.

403 For a description of the SMOLGOL-61 language see: Communications of the Assoc. for Comp. Mach., 1961, Vol. 4, No. 11, pages 499-502.

Chapter 6

PREDICATE CALCULUS AND THE PROBLEM OF AUTOMATION OF THE SCIENTIFIC CREATIVE PROCESSES

§1. BASIC CONCEPTS OF PREDICATE CALCULUS

As we mentioned in Chapter 2, the simples component part of mathematical logic – propositional calculus – does not really penetrate into the structure of the elementary propositions, thereby limiting its capabilities in the formalization of the more complex thought processes. The next higher stage of mathematical logic with regard to complexity, termed restricted predicate calculus or first degree predicate calculus, posseses far stronger expressive capabilities.

One characteristic feature of predicate calculus is, first of all, that along with the variable propositions which can take only two possible values ("true" and "false") there are introduced into consideration the so-called object variables which run through some, generally speaking, infinite region of values, which is customarily termed the object region. The values composing this region are usually termed objects.

Fixing a particular object region, we obtain the possibility of constructing the propositional functions of the object variables, usually termed predicates: the n-place predicate $P(x_1, x_2, \ldots, x_n)$ is a variable proposition whose truth or falsity is determined by sets of values of the object variables $x_1, x_2, \ldots, x_n$. If the predicate P is not identically true or identically false, then on some sets of values of the object variables it takes the value "true" and on others - the value "false."

In the classical theory of predicates only the single-place predicates were called predicates (or properties). For the multiplace predicates the special term "relation" was used: the two-place predicates were termed binary relations the three-place were termed ternary relations, etc. For our purposes there is no need for special emphasis of this difference, therefore we shall term predicates any functions of any (greater than zero) number of object variables.

The use of predicates permits the construction of a formal language analogous to propositional calculus but, in contrast with it, penetrating into the structure of the elementary propositions. For example, the proposition "four is larger than two" is indecomposable in propositional calculus. However, if we introduce the predicate $P(x, y)$ with the set of whole nonnegative numbers as the object region, true if and only if the inequality $x > y$ is satisfied, then this proposition is written in the form $P(4, 2)$, which now gives an idea of the internal structure of the proposition.

The internal structure of the proposition "oxygen is a gas" can be revealed in exactly the same way. To do this it is sufficient to introduce the predicate "is a gas" which takes the value "true if and only if there is substituted in it an object of the object region which actually is a gas. If we designate this predicate by $Q(x)$, then the phrase which we presented can be written in the form Q(oxygen).

In the formal construction of predicate calculus we are not usually interested in the exact objects from which a particular object region is constituted, it is sufficient to know only the number of all these objects or, expressing it more precisely, the power of the set of all the objects composing the object region. If the object region is finite or countable the objects composing it can be replaced by their numbers. Thereby the object regions are reduced to number sets, which facilitates the problem of concrete expression of the corresponding predicates. Since the predicates are variable propositions, all the operations used in the second chapter in the construction of the propositional calculus can be used with them. At the same time the use of the object variables permits the introduction of several new operations which are specific for the predicate calculus. The construction of these operations is accomplished with the aid of the so-called quantifiers. Usually we limit ourselves to only two forms of quantifiers, termed existensional quantifiers and generality quantifiers. For their designation we shall use the symbols $\exists x$ and $\forall x$ respectively, where x indicates the variable on which the quantifier acts.

The expression $\exists x P(x)$ is the conventional designation for the proposition "three exists that object x for which the predicate P is true." Similarly the expression $\forall x P(x)$ designates the proposition "for all objects x the predicate P is true." Here it is understood that the objects under discussion belong to the particular fixed object region M. If the region M consists of the finite number of objects x_1, x_2, ..., x_k, the expression $\exists x P(x)$ reduces to the disjunction $P(x_1) \vee P(x_2) \vee \dots \vee P(x_k)$, and the expression $\forall x P(x)$ reduces to the conjunction $P(x_1) \wedge P(x_2) \wedge \dots \wedge P(x_k)$. In the case of an infinite object region this reduction is not possible, since the constructive nature of our constructions excludes the possibility of the use of infinite disjunctions and conjunctions.

In the nonconstructive (the so-called set-theoretic) approach to the construction of predicate calculus, we can always picture the expressions $\exists x P(x)$ and $\forall x P(x)$ as disjunction and conjunction extended to all the objects x composing the given object region M. In the constructive approach this representation can be used only for the heuristic (inductive) reasonings and constructions, but not at all as a method of strict formal proof.

In the expressions $\exists x P(x)$ and $\forall x P(x)$ the variable <u>x</u> is bound by the corresponding quantifier. In contrast with the <u>free</u> (unbound) variables, for example the variables <u>x</u> and <u>y</u> in the expression Q(x, y), the bound variables do not have independent (individual) value, since the proposition containing the bound variables actually <u>does not depend</u> on these variables. The role of the bound variables in the predicate calculus in this sense is completely analogous to the role played by the variable index <u>i</u> in the calculation of the sum $\sum_{i=1}^{n} f(i)$ or the integration variable <u>x</u> in the calculation of the definite integral $\int_{a}^{b} f(x)dx$. We can, in particular, replace the bound variable with any other variable without altering the sense or value of the corresponding expression in so doing.

Any formula of restricted predicate calculus is constructed with the aid of the four operations of propositional calculus (negation, disjunction, conjunction and implication) and two coupling operations with the aid of the object quantifiers (generality and existensional) from elementary propositions, which are usually the familiar variable propositions (propositional letters) and the propositional functions (predicates) defined above. Here the object region is assumed fixed, and the total number of symbols composing any formula must of necessity be finite. For unity of terminology the elementary variable propositions which do not depend on the object variables (i.e., the propositional letters) are conveniently considered as zero-place predicates (propositional functions of an empty set of object variables).

Just as in propositional calculus, in the predicate calculus we can make use of round brackets for the designation of the order of operations in the formulas. These brackets are also used to establish the action region of the quantifiers which appear in the formula. For example, in the expression $\exists x(P(x,y) \supset Q(x)) \supset R(x)$ the action region of the existensional quantifier $\exists x$ includes only the expression $P(x,y) \supset Q(x)$. The variable x appearing in this expression is bound by the indicated quantifier, while the variable x in the predicate R(x) must be considered as a free variable.

In order to avoid confusion during the various sorts of transformations of formulas in predicate calculus, we usually prefer to redesignate the bound variables so that their notations differ both from one another and from the notations of all the free variables appearing in the same formula. In this case we can consider that the action region of each quantifier extends from the place of its occurrence right to the very end of the formula. Thereby the use of brackets for the designation of action regions can be made superfluous. Hereafter we shall adhere, as a rule, to precisely this interpretation of the action regions of the quantifiers.

We note that in restricted predicate calculus only the object varibles are permitted to be bound using the quantifiers. Here the predicates appearing in a formula are assumed to be unchanging. Such a limitation naturally restricts the region of applications of the logical calculus which we are constructing, which explains the inclusion in its name of the term "restricted." In the so-called extended predicate calculus use is made of variable predicates and predicate quantifiers. In other words, there are permitted expressions of the form "P(x) is valid for every predicate P" or "there exists the predicate Q for which the proposition Q(x) is true" etc. With unlimited use of predicate quantifiers there arises the possibility of construction of internally contradictory formulas and the appearance of paradoxes. All this leads to the necessity for further complication of the corresponding calculi. However, we shall not concern ourselves with a detailed study of the extended predicate calculus, but shall concentrate our attention on the restricted predicate calculus. Therefore hereafter when we use the term predicate calculus (unless otherwise stipulated we shall always mean the restricted predicate calculus. We also shall not consider other possible generalizations of the predicate calculus, for example predicate calculus with several object regions rather than only one, etc.

Just as in the case of propositional calculus, in the construction of predicate calculus it is not sufficient to indicate only the method of writing the formulas. It is necessary also to give the rules for the transformation of the formulas, expressed by axioms. The axioms of predicate calculus 'include all 11 axioms of propositional calculus which were given in §5 of Chapter 2. In addition to them, there are introcuced four postulates which are specific for predicate calculus and to which we assign the numbers from 12 to 15 inclusive:

12. $\dfrac{C \supset P(x)}{C \supset \forall x P(x)}$.

13. $\forall x P(x) \supset P(t)$.

14. $P(t) \supset \exists x P(x)$.

15. $\dfrac{P(x) \supset C}{\exists x P(x) \supset C}$.

With the aid of these postulates (axioms) we can perform the formal deduction of new formulas by exactly the same method as in the case of propositional calculus. We note only that the expressions P(x) and P(t) in axioms 12-15 must be understood not only as elementary one-place predicates, but also as any formulas of predicate calculus containing the letters x and t as free variables. Here it is not excluded that other free variables can appear in the corresponding formulas. In the formal axiomatic construction of predicate calculus we do not usually use the symbols of the individual objects or individual predicates, so that the predicates appearing in the formulas are taken to be any, and not fixed, predicates. In place of the propositional letters A, B, C in the axioms 1-15 there can be substituted any formulas of predicate calculsu, including those which contain free variables.

In all the substitutions which we are discussing here it is understood that the free and bound variables, and also the various bound variables in the formulas obtained as a result of the substitutions, must be designated with different letters. This condition permits avoiding the so-called <u>collision of variables</u>, which leads to unforeseen binding of variables which must be left free. Actually, if, say, in the formula $\exists x P(x) \supset C$ in place of C we substitute the formula Q(x), then the existensional quantifier $\exists x$ would bind not only the variable in the predicate P, but also the variable in the predicate Q. Collision of variables can always be avoided by means of renaming of the bound variables. Hereafter in the case of the necessity of such renaming we shall always assume that it has been accomplished.

Under the condition that the necessary precuationary measures are taken to avoid collision of the variables, all the results on the deducibility of some formulas from others obtained previously in propositional calculus (see the formulas 1-7 in §5 of Chapter 2) are transferred over to predicate calculus. The deduction theorem (with corresponding stipulations) also remains valid in predicate calculus. In particular, if the formula $\underset{\cdot}{B}$ is deducible (in predicate calculus) from the formula $\underset{\cdot}{A}$, then the formula $\underset{\cdot}{A} \supset \underset{\cdot}{B}$ (under the condition of use of measures to prevent occurrence of collision of variables) will be deducible in predicate calculus.

The following rules of deducibility are easily derived from axioms 12-15:

insertion of generality quantifier ($\forall$ -insertion)

$$A(x) \overset{x}{\vdash} \forall x A(x); \qquad (115)$$

insertion of existensional quantifier ($\exists$ -insertion)

$$A(t) \vdash \exists x A(x); \qquad (116)$$

removal of generality quantifier ($\forall$ -removal)

$$\forall x A(x) \vdash A(t); \qquad (117)$$

so-called $\exists$ -removal: if $\Gamma, A(x) \vdash C$, then (see Kleene [42])

$$\Gamma, \exists x A(x) \overset{x}{\vdash} C. \qquad (118)$$

The symbol of the variable x written above the deducibility symbol $\vdash$ means that the corresponding variable is altered (converted from a free variable to an apparent variable) in the process of the deduction in accordance with the axioms (deduction rules) 12 and 15. The free variables which are not altered in the deduction process are customarily termed fixed variables. This last concept can be used for the refinement of the formulation of the deduction theorem.

If there obtains the deducibility of $\Gamma, A \vdash B$, and in the deduction process the free variables occurring in the formula A remain fixed, then there obtains the deducibility of $\Gamma \vdash A \supset B$.

Using the equivalency symbol $\sim$ in the same sense as in propositional calculus, and using A to denote any formula not containing the free variable x, we can easily establish the following relations:

$$\vdash \forall x A \sim A, \quad \vdash \exists x A \sim A; \qquad (119)$$

$$\vdash \forall x \forall y P(x, y) \sim \forall y \forall x P(x, y); \qquad (120)$$

$$\vdash \exists x \exists y P(x, y) \sim \exists y \exists x P(x, y); \qquad (121)$$

$$\vdash \forall x P(x) \supset \exists x P(x); \qquad (122)$$

$$\vdash \exists x \forall y P(x, y) \supset \forall y \exists x P(x, y). \qquad (123)$$

Rules (120) and (121) show the possibility of variation of the order of application of like quantifiers. For the unlike quantifiers this situation does not obtain, since the relation $\vdash \forall x \exists y P(x,y) \supset \exists y \forall x P(x,y)$, dual to the relation (123), in the general case does not obtain in predicate calculus. To convince ourselves of this it is sufficient to consider as the object region the set of all natural numbers, and as the predicate $P(x, y)$ the predicate which is true if and only if $x < y$. Then the formula $\forall x \exists y P(x,y)$ expresses the proposition "for every natural number there exists the natural number y which is larger than x." At the same time the formula $\exists y \forall x P(x,y)$ is the proposition "there exists a natural number which is larger than all the natural numbers." The first prorposition is true and the second is false. Therefore the proposition $\forall x \exists y P(x,y) \supset \supset \exists y \forall x P(x,y)$ with the considered interpretation is false and must not be deducible in a (contensively) consistent calculus.

We take the contensive consistency of predicate calculus (and propositional calculus as well) in the sense that only identically true formulas can be deducible in this calculus, i.e., those formulas which remain true for any object region and for any concrete interpretation of the predicates occurring in them. Without binding ourselves to the requirements of constructivity of arguments (i.e., remaining in the framework of the set-theoretic approach to predicate calculus), it is easy to see that the formulas expressed by axioms 13 and 14, just as the formulas expressed by axioms 1-10 of propositional calculus, are identically true formulas.

The deduction rules (11, 12 and 15) also lead to identically true formulas under the condition of identical truth of their premises. As an example let us consider the deduction rule 12. The identical truth of the premise $C \supset P(x)$ can obtain only in two cases: either when the proposition C is false, or when the proposition $P(x)$ is always true. It is evident that in both of these cases the truth of the proposition $C \dot{\supset} \forall x P(x)$. aldo obtains.

By similar arguments the contensive consistency of predicate calculus is proved (although not completely constructively). The non-structivity which is considered here is associated with the implicitly assumed possibility of the sorting of all the values of the object variables in the determination of theidentical truth, which in the case of an infinite object region requires an infinite number of steps, which is not in agreement with the requirement of finiteness, mandatory for the strictly constructive constructions.

Limiting ourselves to only the finite object regions, it is easy to give a completely constructive nature to the proof of the consistency of predicate calculus. With this limitation the predicate calculus essentially reduces to propositional calculus, since the quantifier binding in this case is simply a short form of writing of the conjunctions and disjunctions extended to all the objects of the object region, and the relations expressed by axioms 12-15 are deducible from axioms 1-11. Thanks to the possibility of such an interpretation, the question on the consistency of predicate calculus reduces to the corresponding question for propositional calculus, which was resolved earlier.

A similar method is used to establish the <u>formal consistency</u> (also termed simple consistency) of predicate calculus, i.e., the impossibility of deduction in this calculus of any formula together with its negation.

The problem of <u>contensive completeness</u> of predicate calculus, i.e., the possibility of formal deduction in this calculus of any identically true formula, was resolved in the positive sense by Godel [16]. It is obvious that in the case of infinite object regions the establishment of the contensive completeness of predicate calculus requires the use of material which goes beyond the limits of finite mathematics. In the case of finite object regions the question on the contensive completeness of predicate calculus reduces to the corresponding question for predicate calculus and therefore is resolved constructively.

In contrast with contensive completeness, also termed completeness in the broad sense, completeness in the narrow sense does not obtain in predicate calculus. Actually, to the list of axioms of predicate calculus there can be adjoined the formula $\exists x P(x) \supset \forall x P(x)$, which is not deducible in this calculus and does not lead to the occurence of a contradiction. The consistency of the axiom system arising as the result of this adjunction becomes clear with consideration of the object region consisting of a single object. In this case the newly adjoined axiom becomes an identically true formula. At the same time, for the object region which now consists of the two objects x and y, this axiom is converted into the formula $P(x) \vee P(y) \supset P(x) \wedge P(y)$, which is not identically true and therefore is not deducible from the remaining axioms.

With the set-theoretic approach to the construction of predicate calculus the following interesting theorem due to Mal'tsev [52] can be proved.

Theorem 1. If an infinite disjunction of (finite) formulas of restricted predicate calculus is an identically true formula, then the finite disjunction of these formulas is identically true.

This theorem can be used successfully for the proof of the so-called local theorems, which in several cases make it possible to transfer to the infinite sets the properties which are valid for all their finite subsets.

Along with the identically true formulas, in predicate calculus it is useful to consider the so-called satisfiable formulas. Satisfiable is the term given to a formula which can be made true with the selection of a suitable object region and a proper definition of the predicates given on it. It is understood that the formulas in question here do not contain symbols of the individual objects or individual predicates.

Every identically true formula is moreover satisfiable, but the reverse is of course not true in the general case. An example of a satisfiable but not identically true formula might be the formula $\exists x P(x) \supset \forall x P(x)$, which was considered above. This formula is identically true only on those object regions which consist of one single object.

It is clear that a formula which is not satisfiable on some object region is identically false on this region. Negations of the identically true formulas. Thereby there is established the connection between the concepts of satisfiability and identical truth of the formulas of predicate calculus.

We can construct examples of formulas which are not satisfiable on any finite object regions, but which are satisfiable on infinite object regions. Moreover the theorem due to Levengeym [Lowenheim] is valid.

Theorem 2. If a formula of predicate calculus is satisfiable on some any infinite object region, then it is also satisfiable on an enumerable object region.

The solvability problem for predicate calculus consists in the indication of a single effective technique (algorithm) for the determination of the satisfiability or nonsatisfiability of any given formula of predicate calculus (on some object region). In contrast with propositional calculus, where the similar algorithm was constructed without any difficulty, the problem of solvability in the general case for predicate calculus, as shown by Church and Turing, in general has no solution. In other words, there does not exist a single constructive technique for the establishment of the satisfiability or the nonsatisfiability of any formula of predicate calculus.

Quite frequently the problem of solvability for predicate calculus is formulated in a somewhat different form: find the algorithm for the determination of the truth (i.e., the identical truth) of any given formula in this calculus.

In view of the contensive completeness of predicate calculus, the algorithm which differentiates the true formulas of the calculus from the false simultaneously solves the problem of the differentiation of the provable and unprovable formulas of this calculus. We note also that from the truth of any formula there follows the nonsatisfiability of its negation. Therefore, if we could decide the question on the satisfiability or nonsatisfiability of all the formulas of predicate calculus we would have the possibility of also resolving the question on the truth of any formula. Unfortunately, in the general case neither the first nor second questions have solutions.

Thus, with respect to the problem of solvability the predicate calculus differs basically from propositional calculus. However, if we limit ourselves to certain particular forms of the formulas the decision algorithm can be constructed in the case of predicate calculus as well.

Such an algorithm can, for example, be constructed for the formulas of predicate calculus which contain only single-place predicates. This situation is the simple result of the fact that for the establishment of the satisfiability or nonsatisfiability of a formula containing $\underline{n}$ single-place predicates it is sufficient to limit ourselves to the consideration of the object regions consisting of no more than 2^n objects. As a result the verification of the satisfiability (or identical truth) of the predicate formula reduces (after replacement of the quantifiers by disjunctions and conjunctions) to the verification of the satisfiability (or, correspondingly, the identical truth) of the corresponding formula of propositional calculus.

In the general case, in the resolution of the question on the satisfiability or nonsatisfiability of any specific formula it may be of considerable assistance to perform a preliminary reduction of this formula to the so-called normal form. We shall differentiate two forms of normal forms: the so-called prenex form and the Skolem normal form. The prenex form is characterized by the fact that all the quantifiers (if there are any must be located at the very beginning of the formula and the action region of each of them must extend to the end of the formula. In the Skolem normal form it is additionally required that all the extensional quantifiers precede all the generality quantifiers.

If the formula is written in the prenex form, then the portion standing after the quantifier (the quantifier-free portion of the formula) can be considered as a formula of propositional calculus (each predicate is considered here simply as a variable proposition). But then we can exclude all the implication signs in this formula (replacing $A \supset B$ by $\neg A \vee B$) and then reduce it to the disjunctive normal form. A similar transformation reduces the original predicate formula to some predicate formula equivalent to it. In many cases the concept of the normal form of the predicate formula includes not only the condition of prenexing of the quantifiers, but also the mandatory reduction of its quantifire-free portion to the ideal disjunctive normal form.

The following theorem is valid.

Theorem 3. For every formula $\underset{\bullet}{A}$ of (restricted) predicate calculus there exists its equivalent formula $\underset{\bullet}{B}$ written prenex form. There exists a single constructive technique (algorithm) for reducing any (predicate) formula to the prenex form.

The validity of the formulated theorem follows from the easily verifiable relations

$$\vdash \neg \forall x P(x) \sim \exists x \neg P(x); \qquad (124)$$

$$\vdash \neg \exists x P(x) \sim \forall x \neg P(x); \qquad (125)$$

$$\vdash Q \wedge \forall x P(x) \sim \forall x (Q \wedge P(x)); \qquad (126)$$

$$\vdash Q \wedge \exists x P(x) \sim \exists x (Q \wedge P(x)); \qquad (127)$$

$$\vdash Q \vee \forall x P(x) \sim \forall x (Q \vee P(x)); \qquad (128)$$

$$\vdash Q \vee \exists x P(x) \sim \exists x (Q \vee P(x)). \qquad (129)$$

Since implication can be replaced by the operations of disjunction and negation, with the aid of the aid of the above formulas with observation of the conditions which exclude the possibility of the occurrence of collision of thevariables, we can perform the sequential permutation of the quantifiers with all the symbols (different from the quantifiers) which make up the formula until all the quantifiers appear in the left part of the formula. For example, the formula $P(x) \vee \neg \forall y Q(x,y)$ can be first transformed to its equivalent formula $P(x) \vee \exists y \neg Q(x,y)$, and then to the (also equivalent) formula $\exists y (P(x) \vee \neg Q(x,y))$, which then is the required prenex form of the original formula.

A direct analogy of theorem 3 does not exist for the Skolem normal form: not every formula of predicate calculus has an equivalent formula having the Skolem normal form.

However the concept of equivalence can be generalized so that any formula of predicate calculus can be reduced to the Skolem normal form. This generalization os given by the concept of the so-called deductive equivalence [61].

The formula $\dot{A}$ is termed deductively equivalent to the formula $\dot{B}$ if by adjoining formula $\dot{A}$ to the axiom set of the calculus we obtain the possibility of deducing the formula B from the thus expanded system of axioms, and, on the other hand, by adjoining formula B to the axiom set we obtain the possibility of deducing formula A

This definition is applicabel not only to predicate calculus, but also to any other logical calculus, in particular to propositional calculus. Since in propositional calculus the adjoining of any nondeducible formula to the axiom set makes all the formulas deducible, then any two nondeducible formulas of predicate calculus are deductively equivalent. It is also clear that any deducible formulas (in any calculus) are deductively equivalent. At the same time, deductive equivalence of the deducible and nondeducible formulas is impossible, since the adjoining of the firs formula to the axiom system does not make the second formula deducible.

Thus, in propositional calculus both all deducible and all nondeducible formulas are deductively equivalent. It is also easy to see that in predicate calculus (as, moreover, in propositional calculus) conventional equivalence of fromulas implies their deductive equivalence. However, the reverse is not true in general, since, for example two elementary propositional variables (arbitrary letters) P and Q, which are not equivalent to one another, are however deductively equivalent.

The following theorem due to Skolem is valid.

Theorem 4. For every formula of (restricted) predicate calculus there exists its deductively equivalent formula written in the Skolem normal form. There exists a single constructive technique (algorithm) which permits performing the reduction of any predicate formula to its deductively equivalent Skolem form.

It can be shown that if two formulas are deductively equivalent, then the identical truth of one of them implies the identical truth of the other. Since there exists a technique for the reduction of any formula of restricted predicate calculus to its deductively equivalent formula in the Skolem normal form, then with the resolution of the problem on the establishment of theidentical truth of particular formulas we can replace these formulas by their corresponding Skolem normal forms. This situation can also be used for the proof of the contensive completeness of predicate calculus, since for that proof it is sufficient, in view of what has been said above, to establish the deducibility of all the identically true formulas written in the Skolem normal form. Actually, by establishing the deducibility of all the indicated formulas we thereby establish the deducibility of all their deductively equivalent formulas, i.e., all the identically true formulas of predicate calculus.

If a formula of predicate calculus contains free variables, it is termed an *open* formula. Formulas which do not contain free variables are customarily termed *closed* formulas. If x_1, x_2, ..., x_n are all the free variables of the open formula $\mathfrak{A}$, then the closed formula $\forall x_1 \forall x_2 \ldots \forall x_n \mathfrak{A}$ is termed the *closure of formula* $\mathfrak{A}$. Any formula $\mathfrak{B}$ is deductively equivalent to its closure $\mathfrak{B}'$ and therefore these two formulas are either simultaneously identically true, or are simultaneously not identically true.

If the problem of solvability is taken in the sense of finding the algorithm which differentiates the true formulas of predicate calculus from the false, then the procedure of closure of the formulas, just as the procedure of reducing the formulas to the normal form (prenex or Skolem) performs the reduction of the general problem of solvability to the corresponding problem for the formulas of some special form. Of course, this reduction does not aid the solution of the problem of the solvability *for all formulas* of restricted predicate calculus. We can, however, identify several quite broad classes of formulas for which decision procedures exist. One class of this kind (formulas containing only single-place predicates) was considered above.

The decidability problem has a positive solution for the case of closed formulas written in the prenex form with either only generality quantifiers (A-formulas) or with only existensional quantifiers (E-formulas). If we denote the number of these quantifiers by m, then the following theorem is valid [1].

Theorem 5. For the close A-formulas with m quantifiers truth need be established only for the object regions which contain no more than m objects. If such of formula is true in the region consisting of m objects then it is an identically true formula.

Theorem 6. A close E-formula is identically true if it is true in the object region containing only one single object. If it is ture in some region, then it is true also in any other region with a larger number of objects.

The decision procedures for the closed A-formulas and E-formulas result directly from these theorems: just as in the case of formulas with single-place quantifiers, the finiteness of the object region permits reduction of the question on the truth of the predicate formulas to the question on the truth of the corresponding formulas of propositional calculus.

The decision problem has a positive solution also for all AE-formulas, i.e., for those closed formulas of restricted predicate calculus in whose prenex normal form all the generality quantifiers precede all the existensional quantifiers (in the Skolem normal form the order of the quantifiers is reversed). All close AEA-formulas are decidable in which the number of existensional quantifiers does not exceed two.

We note that in all cases which we have considered here the decidability was understood in the sense of establishing the truth or falsity of the formulas. With transition to the concept of decidability in the sense of establishing the satisfiability or nonsatisfiability of the formulas, decidable classes of formulas are obtained from the classes (decidable in the first sense) of formulas listed above by the replacement of all the existensional quantifiers by generality quantifiers and vice versa. Thus, for example, the class of all EA-formulas and the class of all EAE-formulas containing no more than two generality quantifiers will be decidable (in the sense of establishing the satisfiability or nonsatisfiability).

A large number of classes of formulas for which the decision problem is resolved positively has now been established. The limitations used to identify the indicated classes concern not only the nature, number and order of arrangement of the quantifiers, but also the form of the quantifier-free parts of the formulas (written in the prenex normal form).

The possibilities have also been investigated of the construction of decision procedures beyond the limits of the restricted predicate calculus, in particular the procedure for the resolution of certain formulas of second degree predicate calculus. In the second degree predicate calculus use is made not only of object quantifiers, but also of predicate quantifiers ("for any predicate P," "there exists the predicate P"), however the predicates can depend only on the object variables and cannot be included in the system of objects composing the object region.

Second degree predicate calculus in the general form not only is not decidable, but also (as shown by Godel) cannot have any complete axiom system. Nevertheless, even in this calculus there exist quite broad decidable parts. Such a part, for example, is the so-called AND-calculus. In this calculus the object region is the set of all natural numbers, and all the predicates are single-place. A corresponding result was announced by Byukh, who somewhat earlier constructed a decision algorithm for a weakened variant of the AND-calculus which has found application in the theory of finite automata.

§2. FORMAL ARITHMETIC AND THE GODEL THEOREM

A formal arithemtic can be constructed on the base of the (restricted) predicate calculus. The objects used for the construction of the formal arithmetic are the whole nonnegative numbers 0,1,2,3,... . On the set of all such numbers there are determined the conventional arithmetic operations of addition and multiplication, and also the operation of direct succession $a' = a + 1$ $(a = 0,1,2,\dots)$. This operation gives a method for unique representation of all the natural numbers: $1 = 0'$, $2 = 1' = 0''$, $3 = 2' = 0'''$ etc.

Arithmetic expressions which are customarily termed measures are composed with the aid of these operations from the whole nonnegative numbers and variables which run through the whole nonnegative values. Examples of such terms are the expressions x, $0'$, $x \cdot y' + a'' \cdot z$. We note that in the case of absence of brackets to determine a particular order of operations, the direct succession operation has the right of priority. After it follows the multiplication operation and then addition. Thus, for example, the expression $x \cdot y' + z'$ must be understood as $((x)\cdot(y')) + (z')$, and not as anything else.

By combining two terms with an equality sing, we obtain a proposition which is true or false depending on whether the indicated equality is true or false. All such propositions constitute the set of so-called elementary formulas of formal arithmetic. If in the terms composing the proposition there are variables, then it (this proposition) will be a predicate which is naturally termed an elementary arithmetic predicate. From such elementary predicates with the aid of the operations of (restricted) predicate calculus (including the operation of quantifier binding) there are constructed more complex arithmetic predicates. All the predicates which can be consteucted in this way are termed Godel arithtmetic predicates.

The formulas of the formal arithmeitc system which we have constructed are limited to the formulas which can be constructed from the elementary formulas with the aid of the operations of (restricted) predicate calculus. The axiom system of the formal arithmetic is obtained by supplementing the axiom system of (restricted) predicate calculsu (axioms 1-15) by the specific arithmetic axioms:

16. $P(0) \wedge \forall x((P(x) \supset P(x')) \supset P(x))$ (axiom of mathematical induction).

17. $a' = b' \supset a = b.$

18. $\neg a' = 0.$

19. $a = b \supset (a = c \supset b = c).$

20. $a = b \supset (a' = b').$

21. $a + 0 = a.$

22. $a + b' = (a + b)'.$

23. $a \cdot 0 = 0.$

24. $a \cdot b' = a \cdot b + a.$

For the proper understanding of these relations it is necessary to note that in order to economize brackets a definite order of priority of operations is established in the formulas of the formal arithmetic system which we have constructed: all arithmetic operations (direct succession, multiplication and addition) have priority over equality, and the latter has priority over all the logical operations.

Having the system of axioms (including the deduction rules 11, 12 and 15) we can transfer to the formal arithmetic the concept of (formal) demonstrability (deducibility) and nondeducibility) of the formulas, and also the concepts of formal deduction, identically true and identically false formulas, etc.

With the aid of these axioms we can in a rigorously formal manner establish also the laws of the arithmetic, such as the commutative and associative laws for addition and multiplication, the distributive law for multiplication with respect to addition, etc. We can prove the validity of the relations $\vdash a+1=a'$, $\vdash a\cdot 0'=a$, $\vdash a+b=0\supset a=0\wedge b=0$, $\vdash a\cdot b=0\supset a=0\vee b=0$ and others.

Using axiom 16 we can derive the following general rule for proof by the method of induction.

Let Γ be a set of formulas of (formal) arithmetic which do not contain the variables x as a free variable, and let P(x) be a formula in which the variable x occurs free. Then, if $\Gamma\vdash P(0)$ and $\Gamma, P(x)\vdash P(x')$ (without alteration of the free variables in P(x)), then $\Gamma\vdash P(x)$.

Continuing the arguments in this fashion, we can in a rigorously formal fashion prove all the basic theormes and justify all the basic proof techniques used in the elementary arithmetic constructed by the contensive method.

The resolution of such basic questions as the consistency and c completeness of the axiom system in the case of the formal arithmetic is much more complex than in restricted predicate calculus. Thus, for the proof of the consistency it is necessary to go beyond the framework of the strictly finite methods. In general, completeness does not obtain for the formal arithmetic system which we have constructed. Moreover, incompleteness is retained for any consistent extension of this system obtained as a result of supplementing the axiom system we have written out with any finite number of compatible (i.e., not leading to a contradiction) new axioms. This is the sense of the celebrated Godel theorem on the incompleteness of the arithmetic, which forced a new look at the entire problem of the substantiation of mathematics and automatization (on the base of complete formalization) of the process of the deduction of new theorems in deductively constructed theories.

In order to clarify the basic idea of the proof of the theorem on the incompleteness of the arithmetic, it is necessary to make a preliminary acquaintance with several concepts and auxiliary results. First of all we must formally define the very concept of completeness.

To establish the class of deducible (demonstrable) formulas of the arithmetic it is sufficient to limit ourselves to the consideration of only the closed formulas. Actually, as a result of the easily verifiable relations $P(x_1,x_2,\ldots, x_n) \vdash \forall x_1 \forall x_2 \ldots \forall x_n P(x_1,x_2, \ldots, x_n), \forall x_1 \forall x_2 \ldots$ $\ldots \forall x_n P(x_1,x_2,\ldots,x_n) \vdash P(x_1,x_2,\ldots,x_n)$ of predicate calculus, from the demonstrability of some formula there follows the demonstrability of its closure and vice versa.

A formal arithmetic system is termed (simply) complete if every closed formula $\underset{\bullet}{A}$ is formally decidable, i.e., if one of the formulas $\underset{\bullet}{A}$ or $\neg \underset{\bullet}{A}$ is a decidable formula.

If for the formula $\underset{\bullet}{A}$ its negation $\neg \underset{\bullet}{A}$ is demonstrable, then the formula $\underset{\bullet}{A}$ itself is termed (formally) *refutable*. *Formal decidability* of a closed formula thus means that this formula is either demonstrable or refutable. Since from the naive contensional point of view the closed form must be either true or false, the condition of its formal decidability is a very natural criterion for the resolution of the question on the completeness of the corresponding formal system. The basic idea of the proof of the theorem on the completeness of the arithmetic consists precisely in the actual construction of the formally undecidable closed form. for this construction we need several auxiliary results, which we shall now consider.

In the contensive sense an *arithmetic predicate* is any (not necessarily constructively defined) predicate on the set of all whole nonnegative numbers. The formal arithmetic system which we have constructed gives a method for the constructive specification of some arithmetic predicates. To establish this method let us introduce the following definition.

The arithmetic predicate $P(x_1,x_2,\dots,x_n)$ is termed numerically expressible if there exists the formula $P_1(x_1,x_2,\dots,x_n)$ of the formal arithmetic system which we are considering, not containing any free variables other than $x_1,x_2,\dots,x_n$) and such that for any concrete set of n whole nonnegative numbers $a_1, a_2,\dots,a_n$ there are satisfied the following conditions:

1) if the proposition $P(a_1,a_2,\dots,a_n)$ is true, then $\vdash P_1(a_1,a_2,\dots,a_n)$;

2) if the proposition $P(a_1,a_2,\dots,a_n)$ is false, then $\vdash \neg P_1(a_1,a_2,\dots,\dots,a_n)$.

In this case we say that the formula $P_1(x_1,x_2,\dots,x_n)$ numerically expresses the predicate $P(x_1,x_2,\dots,x_n)$.

It is easy to show that the arithmetic predicates $x = y$ and $x < y$ are numerically expressed by the formulas $x = y$ and $\exists z(z' + x = y)$ respectively.

The formula $P_1(x_1,x_2,\dots,x_n)$ which we considered in the example just presented is decidable for any concrete set of values of its free variables $x_1,x_2,\dots,x_n$. The formulas having this property are customarily termed <u>numerically decidable</u> formulas. The verification of the truth of the predicate numerically expressible by such a formula, with any set of values of the object variables, can be carried out, in light of what has been said, by the constructive method. In the formal arithmetic system which we have constructed each formula without variables is decidable, and each formula without quantifiers is a numerically decidable formula.

The concept of numerical expressibility can be established not only for the arithmetic predicates, but also for the (contensively defined) arithmetic functions (whose values are the whole nonnegative numbers). We say that the formula $P(x_1,x_2,\ldots,x_n,y)$ of a formal arithmetic system numerically represents the arithmetic function $f(x_1,x_2,\ldots,x_n)$, if for any set of n whole nonnegative numbers the following conditions are satisfied:

$$1)\ \text{if}\quad f(a_1 a_2,\ldots,a_n) = b,\ \text{then}\ \vdash P(a_1,a_2,\ldots,a_n,b);$$

$$2)\ \vdash \exists y(P(a_1,a_2,\ldots,a_n,y) \wedge \forall z(P(a_1,a_2,\ldots,a_n,z) \supset z = y)).$$

The second of these conditions is the conditions is the condition, expressed in predicate calculus language, of the uniqueness of the specification of the function f with the aid of the predicate P.

We note that the possibility of effective specification of the predicates is not necessarily associated with the use of the apparatus of formal arithmetic. Any n-place arithmetic predicate $P(x_1,x_2,\ldots x_n)$ can be specified with the aid of the n-place arithmetic function $\varphi(x_1,x_2,\ldots,x_n)$ which takes the value 0 on all sets of values of the variables $x_1,x_2,\ldots,x_n$ on which the predicate O[sic] is true, and the value 1 on all those sets on which the predicate P is false. This function is termed the representative function of the considered predicate P. A predicate whose representative function is primitive recursive or general revursive is termed respectively a primitive recursive or general recursive predicate.

Godel has established the following result.

Theorem 1. If the arithmetic function $\varphi(x_1,x_2,\ldots,x_n)$ is primitive recursive, then the (n + 1)-place predicate $\varphi(x_1,x_2,\ldots,x_n) = y$ is Godel arithmetic, i.e., expressible by means of formal arithmetic.

From this theorem it follows, in particular, that all primitive recursive predicates are Godel arithmetic predicates.

With the aid of theorem 1 we can easily establish the validity of the following proposition.

Theorem 2. Every primitive recursive predicate is numerically expressible in formal arithmetic.

The following result has been established relative to the general recursive predicates of Kleene [42] and Post.

Theorem 3. With any $n \geq 0$ every general recursive predicate $P(x_1, x_2, \ldots, x_n)$ can be represented both in the form $\exists y R(x_1, x_2, \ldots, \ldots, x_n, y)$ and in the form $\forall y S(x_1, x_2, \ldots, x_n, y)$, where the predicates R and S are primitive recursive. And, on the other hand, every predicate which is representable in each of these two forms is general recursive, and it remains general recursive also in the case when the predicates R and S are not primitive recursive, but only general recursive.

The following theorem due to Kleene [42] is also valid.

Theorem 4. All general recursive predicates are Goldel arithmetic.

For various sorts of complex constructions and proofs in formal arithmetic it is advisable to introduce a special numbering for all its formulas and proofs of these formulas (apparatus of the formal arithmetic system which we have constructed). Such a numeration was proposed by Godel and therefore is termed Godel numeration. There are many different ways of accomplishing this. Let us consider one of them.

Before defining the numbering of the formulas it is advisable to someqhat alter the method of their writing, considering not only the formulas themselves, but also their individual parts as formal objects, termed entities. Among the elementary entities there are, first, all the logical symbols $(\supset, \wedge, \vee, \neg, \forall, \exists)$, the equality symbol (=), the symbols for the arithmetic operations (+, •, '), the zero symbol (0) and the two symbols for the designation of the different object variables (x, |). To the different object variables x,y,x,... there are associated the entities x, (|,x), (|,(|,x)), etc. To the terms and formulas of the form $r+s, r', r=s, A \vee B, A \wedge B, \neg A, \forall u A(u), \exists u A(u)$ there are associated the respective entities $(+,r,s)$, $(',r)$, $(=,r,s)$, $(\vee,A,B)$, $(\wedge,A,B)$, $(\neg A)$, $(\forall,u,A(u))$, $(\exists,u,A(u))$. For simplicity of notations the symbols $r,s,A,B,u,A(u)$ and the entities corresponding ot them are not differentiated here. Using these definitions, we can sequentially, step by step, construct the entities corresponding to the various formulas and their component parts. For example, the formula $\forall x(0'+x=y)$ corresponding to the entity $(\forall,x(=,(+(',0),x),(|,x)))$, the term 0' + x corresponds to the entity (+,(',0),x), the formula x = y corresponds to the entity $(=,x,(|,x))$ etc.

Now let us assign to the elementary entities various odd numbers (the Godel numbers of these entities)

$$\begin{array}{ccccccccccccc} \supset & \wedge & \vee & \neg & \forall & \exists & = & + & \cdot & ' & 0 & x & | \\ 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 & 19 & 21 & 23 & 25 & 27 \end{array}$$

Designating by $p_0, p_1, p_2, \ldots$ the sequence of all prime numbers ($p_0 = 2$, $p_1 = 3$, $p_2 = 5, \ldots$ and etc.) and assuming that the entities $a_0, a_1, \ldots, a_m$ are already associated with the Godel numbers $n_0, n_1, \ldots, n_m$, let us associate the entity $(a_0, a_1, \ldots, a_m)$ with the Godel number $p_0^{n_0} \cdot p_1^{n_1} \ldots p_m^{n_m}$. Associating with each formula of the formal arithmetic system which we are considering the Godel number of its corresponding entity, we obtain the sought Godel numeration for all these formula x = y, to which there corresponds the entity (=, x, (|,x)), has the Godel number $2^{15} \cdot 3^{25} \cdot 5^{2^{27} \cdot 3^{25}}$, to the term 0' + x with its corresponding entity (+,(',0),x) there must be assigned the Godel number $2^{17} \cdot 3 \cdot 2^{21} \cdot 3^{23} \cdot 5^{25}$, etc.

It is easy to see that the Godel number of every nonelementary entity is necessarily even. Keeping this fact in mind, and also the uniqueness of the expansion of every natural number into prime factors, it is not difficult to see that from the Godel number we can uniquely recover its corresponding formula. This means that the correspondence between the formulas of the formal arithmetic system which we are considering and their Godel numbers is one-to-one. Thus, in the case of necessity we can make use of only their Godel numbers rather than the formulas of this system.

By analogy with the Godel numeration of the formulas of the arithmetic, we can introduce the Godel numeration for all possible finite sequences of such formulas, among which there we will be, in particular, theproofs of all the demonstrable arithmetic formulas.

For any whole nonnegative number $\underline{a}$, understood contensively, we shall use $\hat{a}$ to designate the representation of this number in the formal arithmetic ($\hat{a}$ represents the symbol 0 with $\underline{a}$ primes). Fixing some Godel numeration of the arithmetic formulas, we shall for any Godel number $\underline{n}$ use P_n to denote that formula which has the number $\underline{n}$ in our numeration. We identify in the formula P_n the variable $\underline{x}$ (on which the formula actually may not depend), writing the formula $P_n(x)$.

Let us now define the two arithmetic predicates $A(a, b)$ and $B(a, b)$, considering the first predicate to be true if and only if the number $\underline{a}$ is a Godel number of the formula $P_a(x)$ such that the formula $P_a(\hat{a})$ is demonstrable, and the number $\underline{b}$ is the Godel number of some proof of it.

Similarly the predicate $B(a,b)$ is considered true if and only if the number $\underline{a}$ is a Godel number of the formula $P_a(x)$ for which the fromula $P_a(\hat{a})$ is refutable, and the number $\underline{b}$ is a Godel number of the proof of the formula $\neg P_a(\hat{a})$.

Using the theorems formulated above, we can prove the validity of the following important lemma.

Lemma. The arithmetic predicates $A(a, b)$ and $B(a, b)$ which we have defined in the case of the Godel numeration fixed above are numerically expressible in the formal arithmetic system with the axioms

Let us construct the formulas $A_1(a, b)$ and $B_1(a, b)$ which numerically express the predicates $A(a, b)$ and $B(a,b)$ respectively, and let us consider the formula $\forall y \neg A_1(x,y)$. . This formulas has the Godel number p and therefore coincides with the formula which we agreed to denote by $P_p(x)$. Now let us consider the formula $P_p(\mathfrak{p})$ which does not have free variables. This formula, in explicit form represented as $\forall y \neg A_1(\mathfrak{p}, y)$, can be considered as a proposition expressing its intrinsic nondemonstrability. Actually, this proposition is the statement that no number can be a Godel number of the proof of the formula which is obtained from formula $P_p(x)$ as a result of the replacement of x by p. But this replacement is just what transforms the formula $P_p(x)$ into the formula $\forall y \neg A_1(\mathfrak{p}, y)$ which we have constructed.

It is found that with some additional assumptions these properties of the formula $\forall y \neg A_1(\mathfrak{p}, y)$ imply its undecidability, which then proves the incompleteness of the formal arithmetic system which we have consteucted. The additional assumptions involved here amount to the fact that the formal arithmetic is assumed to be ω-consistent.

By ω-consistency of the formal arithmetic system we mean the following property: for no formula $P(x)$ for which the formula $\neg \forall x P(x)$ is demonstrable can it be shown that all formulas of the form $P(0)$, $P(1)$, $P(2)$,... . are demonstrable.

From the ω-consistency of the formal arithmetic there results its simple consistency. Actually, let P be any demonstrable formula not containing free variables (for example, the formula $0 = 0$). Introducing into this formula the dummy variable x, on which P actually does not depend, we write it in the form $P(x)$. Then all the formulas $P(0)$, $P(1)$,... coincide with P and, consequently, are demonstrable. As a result of the ω-consistency of the system this means that the formula

$\neg\forall x P(x)$, actually coinciding with the formula $\neg P$, is nondemonstrable. However, with the existence of contradiction in the sys em, thanks to the property of weak $\neg$-removal (see formula (7) of §5 Chapter 2) all the formulas of this system would be demonstrable. Since, however, the formula $\neg P$ is nondemonstrable, our system is (simply) consistent.

Now we can prove the Godel theorem on the incompleteness of the arithmetic in the primitive (weak) form.

Theorem 5. If the formal arithmetic is ω-consistent then formula $\forall y \neg A_1(\hat{p}, y)$, constructed above is an example of a nondecidable formula.

Proof. Let us assume first that the formula $\forall y \neg A_1(\hat{p}, y)$ is demonstrable. We use k to denote the Godel number of the proof of this formula. Then the proposition A(p, k) is true and, consequently, the formula $A_1(\hat{p}, \hat{k})$ is deducible. Using the operation of $\exists$-insertion (see §1 of present chapter) we obtain $\vdash \exists y A_1(\hat{p}, y)$, or, using formula (125), we obtain $\vdash \neg \forall y \neg A_1(\hat{p}, y)$. But then, as a result of the assumption made above, the formal arithmetic system which we are considering will be (simply) inconsistent, which is excluded in view of the condition of its ω-consistency.

Let us now assume that the formula $\neg\forall y \neg A_1(\hat{p}, y)$ is demonstrable. As has been proved, the formula $\forall y \neg A_1(\hat{p}, y)$ is nondemonstrable. Therefore none of the numbers 0,1,2,... is the Godel number of the proof of the latter formula. This means that all the propositions A(p, 0), a(p, 1), A(p, 2),... are false, and consequently, in view of the numerical expressibility of the predicate A, all the formulas of the form $\neg A_1(p,i)$ are deducible for i = 0,1,2,... . But then, on the strength on the assumption of the ω-consistency of the system, the formula $\neg\forall y \neg A_1(\hat{p}, y)$ is nondemonstrable, which contradicts the assumption we made on its demonstrability.

Thus, the formula $\forall y \neg V_1(\hat{p}, y)$ cannot be either a demonstrable or a nondemonstrable formula and, consequently, is an example of the non-decidable formula. Thereby the theorem is proved.

As we see from this proof, for the establishing of the nondemonstrability of the formula which we have constructed the assumption on the simple consistency of the formal arithmetic system is sufficient, and the assumption on the ω-consistency of the system is used only for the proof of the nonrefutability of this formula.

Rosser has shown [71] that we can construct an example of a formula whose nondecidability is established without the assumption on the ω-consistency of the formal arithmetic. Its simple consistency is sufficient for this. The formula involved here is constructed as follows. First from the predicates A(a, b) and B(a, b) defined above we construct the formula $\forall y(\neg A_1(x,y) \vee \exists z(z<y \wedge B_1(x, z)))$(where the formulas A_1 and B_1 numerically express the predicates A and B respectively). If we designate this formula by $P_q(x)$ (q is its Godel number) then the formula $P_q(\hat{q})$ is the desired example of a formula whose nondecidability is established with the aid of the assumption on the simple consistency of the formal arithmetic. The validity of this last assumption has been established by Ackerman, Neyman with a certain limitation, and by Gentzen in the general case.

Novikov [59] has shown not only the simple consistency but even the ω-consistency of the arithmetic, although to do this required resort to methods going beyond the framework of the formal arithmetic itself. From this result and theorem 5 there follows the Godel theorem on the incompleteness of the arithmetic:

<u>Theorem 6</u>. The formal arithmetic system with the axioms 1-24 is incomplete in the sense that in it there are constant propositions (formulas which do not contain free variables) which cannot be proved or refuted using the apparatus of this system.

We might think that the Godel result uncovers only the insufficient completeness of our selected axiom system for the formal arithmetic, and that with suitable supplementing of this system of axioms by new axioms the incompleteness of the arithmetic (while retaining its consistency) would no longer obtain. In actuality the matter is far from geing this simple. As shown by the detailed analysis carried out by Godel, with any consistent extension of the axiom system the formal arithmetic continues to remain incomplete, and just as before there will be in it nondecidable closed formulas. Moreover, every formal system which satisfies certain quite general conditions (the existence of a sufficiently extensive set of formulas and objects), in case of its consistency will of necessity be incomplete.

As mentioned above, the proof of the consistency of the formal arithmetic system S which we have constructed required resort to apparatus which goes beyond the framework of this system. It is found that this fact is not chance: it can be shown that the proof of the consistency of the system S by the apparatus formalized in this very system is not possible.

Actually, in the system S it is found to be impossible to prove the formula 1 = 0. In the case of the inconsistency of this system, all its formulas and, in particular the formula 1 = 0, become demonstrable. The reverse is also true: from the demonstrability of the formula 1 = 0 there follows as a corollary the inconsistency of the system S. Let $\underline{r}$ be the Godel number of the formula 1 = 0. Then the formula $\neg\exists y A_1(\hat{r}, y)$, which for brevity we denote by $\underset{\bullet}{A}$, as a result of the definition presented above of the predicate A(x, y) with the numerical expression $A_1(x, y)$, is the formal expression of the consistency of the system S.

It can be shown that the formalization (in the system S) of the proof of theorem 5 can be reduced to the deduction (in S) of the formula $\mathfrak{A} \supset \forall y \neg A_1(\bar{p}, y)$, and the formalization (in S) of the proof of the consistency of the system S can be re uced to the deduction (in S) of the formula $\underset{\cdot}{A}$. But in the case of the existence of both of the indicated deductions, according to the deduction rule expressed by axiom 11 of propositional calculus (see Chapter 2 §5), the formula $\forall y \neg A_1(\bar{p}, y)$ must also be deducible (demonstrable). Since this contradicts theorem 5, then formula $\underset{\cdot}{A}$ cannot be demonstrable in the system S, which then shows the impossibility of the proof of the consistency of the formal arithmetic system using the apparatus of this system itself.

§3. CONCEPT OF AUTOMATION OF PROOFS AND CONSTRUCTION OF DEDUCTIGE THEORIES

The formal arithmetic system constructed in the preceding chapter is an example of the formalization of the mathematical theory on the basis of the predicate calculus. Such formalization makes it possible to expand into exactly defined elementary component parts the process of the proof of all the propositions which are demonstrable in the framework of the given theory. By placing in the program of a universal electronic digital machine all the axioms and derivation rules of the considered theory, and also the formula expressing the proposition which is to be proved, we can organize a system of random search for the proof of this formula.

If the number of elementary steps which permit accomplishing the proof of the required formula is relatively small, then the high speed of operation of the electronic digital machine permits finding the proof by the method of simple sorting of all the rariants. However, for any complex propositions such a method of search for the proof becomes unsuitable in practice inview of the fact that the number of variants to be sorted becomes tremendously large, so that their complete sorting in a reasonable time is not possible even on the modern high-speed electronic digital machines. In these conditions we must make use of various sorts of techniques which permit a sharp reduction of the number of variants to be sorted. Such techniques include the enlarging of the deduction rules, thanks to which the proof is constructed from larger blocks and as a result becomes considerably shorter. Another technique for the shortening of the sorting consists in the development of various heuristic methods which make it possible to set intermediate goals and thereby break the proof search process down into individual stages. Such stages must be small enough so that complete sorting within them is possible.

Usually in the automation of proofs we prefer to make use of a formalization system of the predicate calculus which is somewhat different from that which was developed in the first section of the present chapter. Gentzen proposed such a system of formalization in [15]. It permits the normalization in some sense of the process of the proof, using the formal apparatus of the predicate calculus. We shall present certain basic these of the Gentzen system of formalization of predicate calculus, which, in contrast with the previously considered so-called Hilbert system H, we shall designate by G or, more precisely, by G1.

One of the significant concepts in the Gentzen system is the concept of the so-called <u>sequence</u>. A sequence is a formal expression of the form $A_1, A_2, \ldots, A_m \to B_1, B_2, \ldots, B_n$, where A_i and B_j are formulas, and the arrow denotes a new formal symbol. The sequence $\mathfrak{A}_1, \mathfrak{A}_2, \ldots, \mathfrak{A}_m \to$ $\to \mathfrak{B}_1, \mathfrak{B}_2, \ldots, \mathfrak{B}_n$ has the same interpretation as the formula $\mathfrak{A}_1 \wedge \mathfrak{A}_2 \wedge \ldots \wedge \mathfrak{A}_m \supset$ $\supset \mathfrak{B}_1 \vee \mathfrak{B}_2 \vee \ldots \vee \mathfrak{B}_n$ in the Hilbert system, where the conjunction of an empty set of formulas is considered true, and the disj8nction of an empty set of formulas is considered false.

The part of the sequence standing ot the left of the symbol $\rightarrow$ is termed the <u>antecedent</u>, and the part standing to the right of this symbol is termed the <u>succeedent</u> of the considered sequence. For brevity of writing, the finite sequences of formulas are denoted by the capital Greek letters ($\Gamma, \Theta, \Delta, \Lambda$) etc.) and the individual formulas are denoted by the capital Latin letters.

In the Gentzen system Gl there is the natural axiom (axiom scheme)

$$C \rightarrow C, \qquad (130)$$

and also a whole series of <u>deduction rules</u> which are divided into rules of deduction for propositional calculus and additional rules for deduction for predicate calculus.

The deduction rules for propositional calculus:

$$\frac{A, \Gamma \rightarrow \Theta, B}{\Gamma \rightarrow \Theta, A \supset B} \text{ ($\supset$-insertion in succeedent)} \qquad (131)$$

$$\frac{\Delta \rightarrow \Lambda, A \quad B, \Gamma \rightarrow \Theta}{A \supset B, \Delta, \ \Gamma \rightarrow \Lambda, \Theta} \text{ ($\supset$-insertion in antecedent)} \qquad (132)$$

$$\frac{\Gamma \rightarrow \Theta, A \quad \Gamma \rightarrow \Theta, B}{\Gamma \rightarrow \Theta, \ A \wedge B} \text{ ($\wedge$ -insetion in succeedent)} \qquad (133)$$

$$\frac{A, \Gamma \rightarrow \Theta}{A \wedge B, \Gamma \rightarrow \Theta} \quad \frac{B, \Gamma \rightarrow \Theta}{A \wedge B, \Gamma \rightarrow \Theta} \text{ ($\wedge$ -insertion in antecedent)} \qquad (134)$$

$$\frac{\Gamma \rightarrow \Theta, A}{\Gamma \rightarrow \Theta, A \vee B} \quad \frac{\Gamma \rightarrow \Theta, B}{\Gamma \rightarrow \Theta, A \vee B} \text{ ($\vee$. -insertion in succeedent)} \qquad (135)$$

$$\frac{A, \Gamma \rightarrow \Theta \quad B, \Gamma \rightarrow \Theta}{A \vee B, \Gamma \rightarrow \Theta} \text{ ($\vee$ -insertion in antecedent)} \qquad (136)$$

$$\frac{A, \Gamma \rightarrow \Theta}{\Gamma \rightarrow \Theta, \neg A} \text{ ($\neg$ -insertion in succedent)} \qquad (137)$$

$$\frac{\Gamma \rightarrow \Theta, A}{\neg A, \Gamma \rightarrow \Theta} \text{ ($\neg$ -insertion in antecedent)} \qquad (138)$$

Additional rules of deduction for predicate calculus:

$$\frac{\Gamma \rightarrow \Theta, A(b)}{\Gamma \rightarrow \Theta, \forall xA(x)} \text{ ($\forall$ -insertion in succeedent)} \qquad (139)$$

$$\frac{A(t), \Gamma \rightarrow \Theta}{\forall xA(x), \Gamma \rightarrow \Theta} \text{ ($\forall$ -insertion in antecedent)} \qquad (140)$$

$$\frac{\Gamma \rightarrow \Theta, A(t)}{\Gamma \rightarrow \Theta, \exists xA(x)} \text{ ($\exists$ -insertion in succeedent(} \qquad (141)$$

$$\frac{A(b), \Gamma \rightarrow \Theta}{\exists xA(x), \Gamma \rightarrow \Theta} \text{ ($\exists$ -insertion in antecedent)} \qquad (142)$$

In rules (139) and (142) there must be observed a definite limitation which amounts to the following: the variable b must not occur free in the conclusions (i.e., in the expressions under the bar) or (139) and (142).

We note that when formula A(x) does not actually contain the free variable x, then A(b) coincides with A(x). In this case the variable b can be arbitrary so that as b we can always select a variable which does not occur in the conclusion and can thereby observe the required limitation.

In addition to the rules above, in the Gentzen system there are seven more so-called structural rules of deduction:

$$\frac{\Gamma \to \Theta}{\Gamma \to \Theta, C} \text{ (refinement in succeedent)} \qquad (143)$$

$$\frac{\Gamma \to \Theta}{\Gamma, C \to \Theta} \text{ (refinement in antecedent)} \qquad (144)$$

$$\frac{\Gamma \to \Theta, C, C}{\Gamma \to \Theta, C} \text{ (abbreviation in succeedent)} \qquad (145)$$

$$\frac{C, C, \Gamma \to \Theta}{C, \Gamma \to \Theta} \text{ (abbreviation in antecedent)} \qquad (146)$$

$$\frac{\Gamma \to \Lambda, C, D, \Theta}{\Gamma \to \Lambda, D, C, \Theta} \text{ (permutation in succeedent)} \qquad (147)$$

$$\frac{\Delta, C, D, \Gamma \to \Theta}{\Delta, D, C, \Gamma \to \Theta} \text{ (permutation in antecedent)} \qquad (148)$$

$$\frac{\Delta \to \Lambda, C \quad C, \Gamma \to \Theta}{\Delta, \Gamma \to \Lambda, \Theta} \text{ (section)} \qquad (149)$$

For the designation of the demonstrability of the sequence S in the system G1 use is made of the abbreviated notation $\vdash$ S, similar to the corresponding notation in the Hilbert system H.

The Gentzen system G1 is in a certain sense of the word equivalent to the Hilbert system H, since, as shown by Gentezen, the following theorem is valid:

Theorem 1. If the formula A is deducible from the finite set of formulas Γ in the Hilbert system H and all the variables remain fixed, then in the Gentzen system G1 the sequence Γ → A is deducible. And, on the other hand, if in the system G1 the sequence $\Gamma \rightarrow A$ is deducible, then the formula A is deducible from the set of formulas Γ and in this case all the variables remain fixed.

The similarity between the Hilbert and the Gentzen systems is so great that if in the deduction performed in one system use is made of the rules (axioms) only for a part of the logical operations $\subset, \neg, \vee, \wedge, \forall, \exists$, then in the corresponding deduction in the other system we could be limited to only the rules with the same symbols, with the possible exception of the implication symbol $\supset$.

Gentzen established a result which makes it possible to eliminate from the proofs in the system G1 the use of the sections (deduction rule (149)). This is the so-called Gentzen theorem on the normal form, or the elimination theorem.

Theorem 2. Let in the system G1 there be given the proof of some sequence in which no variable occurs free and bound simultaneously. Then in G1 there is a proof of the same sequence which does not use the sections (rule (149)) and uses only the logical rules which were used in the orginal proof.

Along with the system G1, Gentzen has also constructed other formal systems (the systems G2, G3).

Hao-Wang [77] has used the Gentzen system G1 for the automation (with the aid of a universal electronic digital machine) of the process of the proof of a large number of theorems not only from propositional calculus, but also from the (restricted) predicate calculus. The experiments made by Hao-Wang showed that in spite of the absence of a universal decision procedure for the predicate calculus, we can construct a partial decision procedure which permits proof of all the theorems usually included in a handbook on mathematical logic.

In the case of propositional calculus, for the proof of a particular sequence the deduction rules (131)-(138) of the Gentzen system (supplemented by the rules for the insertion of the equivalence symbol into the succeedent and the antecedent) are used by Hao-Wang in the reverse direction (the conclusion is replaced by the premise). In this case there is performed a sequential (beginning with the left end of the sequence) exclusion of the logical connections. As a result of the application of this procedure, after a finite number of steps we obtain the sequence of the form $A_1, A_2, \ldots, A_m \rightarrow B_1, B_2, \ldots, B_n$, where A_i and B_j are the so-called atomic formulas, i.e., simply speaking, the propositional letters. Similar sequences, which are naturally termed elementary, are demonsteable if and only if in their left and right parts there is encountered the same atomic formula.

If all the elementary sequences obtained as a result of this procedure are demonstrable, then the original sequence is obviously demonstrable. For its proof it is sufficient to repeat all the steps which led to the appearance of the indicated elementary sequences, in the reverse order.

If the theorem to be proved is written in the form of a formula in the Hilbert system H, then for converting it to a sequence it is sufficient to place an arrow in front of it. If the last operation performed in the original formula is implication, the formula can be converted to a sequence by replacing the corresponding symbol $\supset$ by an arrow. This method of converting the formula to a sequence usually leads to a shorter proof than with the writing of the arrow in front of the formula. As a result of the definition of the meaning of the symbol $\rightarrow$ in the sequence and theorem 1 of the present section, the proof of the sequence obtained by either of the two indicated methods is also the proof of the original formula.

Let us consider as an exampoe the formula $\neg(A \vee B) \supset \neg A$ of propositional calculus. Replacing the implication symbol by the arrow, we convert it to the sequence $\neg(A \vee B) \rightarrow \neg A$. The extreme left logical connective is the negation symbol $\neg$. Reversal of the rule for $\neg$-insertion into the antecedent (rule (138)) brings our sequence to the form $\rightarrow \neg A, A \vee B$. Elimination of the following logical connective (which is again negation) leads (with the aid of the reversal of rules (147) and (137)) to the sequence $A \rightarrow A \vee B$. Finally, reversal of rule (135) brings our sequence to the form $A \rightarrow A, B$, which is an elementary sequence. Since the letter A occurs in both the left and right parts of the last sequence the sequence is demonstrable. Writing out in the reverse order all the steps which led us to the sequence $A \rightarrow A, B$, we come to the proof of the original sequence $\neg(A \vee B) \rightarrow \neg A$.

For the proper understanding of the last step in the described example of sequential elimination of the logical connectives, we note that the deduction rule (135) can (as Hao-Wang does) be written

$$\frac{\Gamma \rightarrow \Theta, A, B}{\Gamma \rightarrow \Theta, A \vee B}. \qquad (150)$$

Similarly in rule (134) for $\wedge$-insertion into the antecedent the two premises can be replaced by one premise of the form $\Gamma, A, B, \rightarrow \Theta$ The legitimacy of these changes of the rules (134) and (135) is easily justified with the aid of the rules for refinement in the succeedent and antecedent (rules (143) and (144)).

Of course, for the propositional calculus we can construct more effective proof procedures, however the described proof is good in that it permits generalization to the case of predicate calculus. In this generalization use is made of the technique of elimination of the quantifiers with the aid of the reversal of the deduction rules (139)-(142), completely analogous to the technique described above for the elimination of the logical connectives with the aid of the reversal of the reversal of deduction rules (131)-(138).

The decision procedure constructed by Hao-Wang encompasses, naturally, only a part of the formulas of predicate calculus (since a decision procedure does not exist for predicate calculus as a whole). However, it is sufficient to cover almost all the theorems of predicate calculus included in such a major monograph as Whitehead and Russell's Principia Mathematica.

Improvement of the effectiveness of the decision procedure is achieved by means of the use of several additional technique. Among these techniques an important place is occupied by the reduction of the formulas to the so-called minisphere form. In contrast with the prenex form in which the region of action of the quantifiers is the maximum possible, the minisphere form of the formulas provides for the greatest possible reduction of the region of action of the quanfifiers. In the case of the reduction of the formulas to the minisphere form, the operations of implication and equivalence are usually first expressed by means of the operations of disjunction, conjunction and negation. In this case the concept of the minisphere form can be refined by means of the following operations.

First, the individual propositional letters and elementary predicates are minisphere formulas. Second, if the formulas A and B have the minisphere form, then the formulas $A \vee B$, $A \wedge B$ and $\neg A$ also are minispheric. Third, if P(x) is a disjunction (or, respectively, a conjunction) or minispheric formulas, then the formula $\forall x P(x)$ (or, correspondingly, the formula $\exists x P(x)$) will also have the minisphere form. Fourth, if the formula P(x) in $\forall x P(x)$ (or Q(x) in $\exists x Q(x)$) begins with an existensional quantifier (or, respectively, with a generality quantifier) and the formula P(x) (and Q(x)) is minispheric, then the formula $\forall x P(x)$ (and $\exists x Q(x)$) will also be minispheric. Finally, fifth, a formula which begins with a chain of like quantifiers has the minispheric form if every formula obtained from it by permutations of these quantifiers and dropping the first of them has the minispheric form.

The procedure for reduction of the formulas of predicate calculus to the minisphere form is frequently quite simple. In this case it is advisable to begin the decision procedure with the reduction of both parts of the given sequence to the minisphere form and with simultaneous elimination (wherever this is possible) of all the logical connectives with the aid of the reversion of the deduction rules (131)-(138).

For the elimination of the quantifiers, in place of the application of the (reversed) deduction rules (139)-(142) which requires certain limitations, it is frequently advisable to use a simpler method based on the concept on the signs of the quantifiers occurring in the particular sequence. In the definition of this concept we first consider the question on the assignment of signs to the various parts of the formulas of predicate calculus. First of all, each formula, considered as an occurrence in itself, is regarded as positive. If P is a positive (negative) part of the formula Q or the formula R, then P will be a positive (or, correspondingly, negative) part in the formulas $Q \wedge R$, $Q \vee R$, $\forall x Q$, $\exists x Q$. If D is a positive (negative) part in the formula S, then D will be a negative (positive, respectively) part of the formulas $\neg S$ and $S \supset Q$, while D will be a positive (negative, respectively) part of the formula $Q \supset S$.

If a part of any formula from the set of formulas composing a sequence is considered a part of the sequence, then any part in the sequence will have the same sign as in the corresponding formula if this formula occurs in the succeedent, and the opposite sign if this formula occurs in the antecedent of the considered sequence. Every generality quantifier in the sequence is assigned that sign which its action region has in this sequence (considering the action region as a part of the sequence). The signs of the existensional quantifiers are considered to be opposite to the signa of their action regions. For example, in the sequence $\forall x \exists y P(x,y),\ \neg \forall v Q(v) \rightarrow \forall z R(z) \wedge \exists u S(u)$ the quantifiers $\forall z$, $\forall v$ and $\exists y$ are positive, while the quantifiers $\forall x$ and $\exists u$ are negative. In establishing the signs of the quantifiers it is necessary that all the variables bound by the quantifiers be pairwise different. Their notations must also differ from the notations of all the free variables. With satisfaction of these conditions, the following decision procedure can be constructed for the sequences which are in the AE-form, i.e., consist of formulas in which no existensional quantifier can include in its action region generality quantifiers.

First, all the formulas occurring in the sequence are reduced to the minisphere form. Then with the aid of the reversion of rules (131) -(138) we eliminate all the logical (propositional) connectives which permit such elimination. The resulting sequences must be in the AE-form (since otherwise the original sequence would not be an AE-sequence). In all these sequences all the quantifiers are omitted, the variables bound by the negative quantifiers are replaced by pairwise different numbers, and the vairables bound by the positive quantifiers are retained without change.

Again applying the reversion of the rules (131)-(138), we reduce the resulting sequences to the elementary form, i.e., to the form not containing either quantifiers or propositional logical connectives. The true elementary sequences (i.e., those sequences in which there is at least formula common to the antecedent and the succeedent) are thrown out. Performing all possible (not necessarily one-to-one) substitutions of variables in place of the numbers in the remaining sequence, we attempt to make all of them true. If this can be done, then the original sequence was true, if not, then it was false.

Let us consider as an example the two sequences: $\forall x P(x) \rightarrow \exists y P(y)$ and $E x P(x) \rightarrow \forall y P(y)$. In the first sequence both quantifiers are negative. Therefore the described procedure for the removal of the quantifiers reduces it to the form $P(1) \rightarrow P(2)$. Performing the substitution of the variable $\underline{x}$ in place of the numbers 1 and 2, we transform the latter sequence to $P(x) \rightarrow P(x)$. Consequently, the first of the initially given sequences is true. Both quantifiers of the second sequence are positive. The procedure for the elimination of quantifiers reduces it to the form $P(x) \rightarrow P(y)$, which in the general case (for any predicate P and a nontrivial object region) is a false sequence. Consequently, the second of theoriginal sequences is false.

These results coincide with the results of the direct verification of the given sequences, which is not difficult to accomplish in this case. We note that if, in spite of the condition stipulated above in the second sequence both bound variables were designated with the same letter, we would come to an incorrect conclusion, taking the sequence to be true. It is also useful to note that the described procedure, even without the preliminary reduction of the formulas to the minisphere form, is suitable for the resolution of all AE-sequences containing no more than one positive quantifier.

Along with the decision procedure described above for the propositional calculus (procedure I), the procedure just described without the reduction of the formulas to the minisphere form (procedure II) was programmed by Hao-Wang for the IBM-704 universal electronic digital machine.

Using program I, the machine required about three minutes to prove all 220 theorems of propositional calculus composing the first five chapters of the monograph Principia Mathematica. The total machine operating time (with account for the time for entry of data and removal of results) amounted to about 37 minutes. Using program II, after about an hour of operation the machine had proved about 130 theorems of predicate calculus from the 158 theorems constituting the following five chapters of the same monograph. In all, program II was able to prove 139 theorems, although the decision time increased considerably to do this.

If we supplement procedure II with the technique for the elimination of quantifiers on the basis of the reversion of rules (139)-(142) and introduce into it certain preliminary transformations of the formulas whic consititute the original sequence, then all 158 of the theorems indicated above become demonstrable. The preliminary transformations involved here amount to the application (as long as possible) of the following replacement rules to the formulas which make up the original sequence:

$\forall x(P(x) \wedge Q(x))$ is replaced by $\forall x P(x) \wedge \forall x Q(x)$; (151)

$\exists x(P(x) \vee Q(x))$ is replaced by $\exists x P(x) \vee \exists x Q(x)$; (152)

$\forall x(P(x) \supset (Q(x) \wedge R(x)))$ is replaced by $\forall x(P(x) \supset Q(x)) \wedge$

$\wedge \forall x(P(x) \supset R(x))$. (153)

These rules to a certain degree replace the procedure for reduction of the formulas to the minisphere form, which in the general case is quite complex. If after the application of these rules and the elimination of the logical connectives with the aid of reversion of rules (131)-(138) all the resulting sequences are AE-sequences and in addition are either minispheric or contain no more than one positive quantifier, then solution of the sequence can, as a rule, be carried out by procedure II.

Hao-Wang also proposed further improvements of the described procedures which make it possible to go beyond the limits of just the AE-formulas. We note that with the aid of one of these improved procedures the IBM-704 machine carried out the proof of 350 theorems from the first nine chapters of Principia Mathematica in 8.5 minutes. The procedures constructed by Hao-Wang can apparently be easily transformed into quasi-decision procedures for the entire restricted predicate calculus in the sense that they can (after suitable complementing) prove any demonstrable formula of this calculus and can refute "almost all" the nondemonstrable formulas. The expression "almost all" is understood here in the quite practical sense and cannot, of course, be understood as "all, except for a finite number."

We should underscore the difference between the purely theoretical and practical approaches to the solution of the problem of decidability. In the theoretical aspect the prime importance lies in the very fact of the existence or nonexistence of the decision procedure for a particular class of formulas. The decision procedures which are constructed for this purpose are in the majority of cases completely unsuitable for the automation of the proofs of the theorems, since they lead to excessively cumbersome and lengthy constructions.

On the other hand, in the practical approach to the construction of the decision procedures particular attention is devoted to the questions of the speed and ease of performance of these procedures. At the same time we frequently reconcile ourselves to the fact that the constructed decision procedure does not encompass absolutely all the formulas of the given class, if with its practical application the cases when it does not give an answer (after some predetermined time) are relatively infrequent. Thus, the practical decision procedures may not be in the exact sense of the word decision procedures, but only quasi-decision procedures.

Therefore it is not surprising that in practice effective decision procedures can be constructed not only in decidable theories, but also in undecidable theories. We shouldnot forget that the human being working in a region of some undecidable theory (for example, in the arithmetic of the natural numbers) makes use of a finite (and, frequently not even very large) number of techniques for the performance of the proofs and the construction of counter-examples. The task of the practical decision procedures is then to formalize these techniques.

Of course, the solution of this problem is simplified if the region of application of the decision procedure is limited ahead of time to some sufficiently narrow region. At the same time the preliminary establishing of the theoretical possibility of the solution of the problem of decidability in the corresponding region, generally speaking, does not simplify the problem of the construction of the practical decision algorithm.

Several decision procedures have been constructed for the relatively simple branches of mathematics (algebra of real polynomials , elementary geometry, theory of Abelian groups with a finite number of generatrices, etc.). However, these procedures were constructed, as a rule, in the purely theoretical aspect, and a considerable amount of effort will be required to transform them into practical decision algorithms.

Of great interest is the problem of the construction of algorithms which would not simply prove or disprove the propositions specified by the human but would themselves search out new interesting theorems in a particular field. For the construction of this sort of algorithm it is necessary to develop sufficiently good criteria for the evaluation of the degree of nontriviality of a theorem.

One of the first attempts in this direction was made by Hao-Wang [77], who constructed a program for the screening (with subsequent proof) of theorems in propositional calculus. This attempt, however, was not completely successful, in view of the paucity of the nontriviality criteria included in the program: the machine printed out too large a number of theorems without performing adequate screening of the uninteresting (trivial) theorems.

The first nontriviality criterion which usually comes to mind is that the nontrivial theorem must be relatively well formulated (be expressed by a short formula) and still not have short proofs. The establishment of still more natural criteria (in agreement with the conventional ideas on the nontriviality of theorems) becomes possible if the process of the screening of new theorems and their proofs is con structe on the principles of self-organization. This can be achieved by means of supplementing the original axiom system by nontrivial theorems selected by the program. It is natural to evaluate the complexity of a theorem on the basis of the minimal number of steps with which its proof can be accomplished. We term the original axioms and all the theorems whose complexity exceeds some threshold which is selected in advance nontrivial propositions. Each newly proved nontrivial proposition is adjoined to the axiom system, with the result that a reevaluation is made of the complexity of all the previously obtained theorems. Excluding from the axiom system the theorems which have become trivial, we look for a new nontrivial theorem, adjoin it to the axiom system, again exclude theorems which have become trivial, etc.

This self-organizing system for the construction of formal deductive theories resembles to a considerable degree the process of the construction of such theories by the human. We should note that the transition to the processes of the construction of the deductive theories on the self-organization principles forces a new approach to the problem of their decidability. Of course, if the indicated process goes on isolated from the outside world, then it is in the final analysis equivalent to some "rigid" (non-self-organizing) algorithm, so that in the formulation of the decidability problem actually nothing is changed. The same will obviously be true in the case when the process is influenced by some algorithm which is external to it (the case of the "constructive external world").

In the case of the "nonconstructive external world" when the external actions on the process which we are considering cannot be reduced to an algorithm, the situation is altered in principle. Actually, let us assume that the process in question can accumulate information coming from the outside and can perform the comparison of it with the formulas of the restricted predicate calculus which it has been given. Let us assume further that the information arriving from the outside consists of two sequences of formulas of restricted predicate calculus, arranged in the order of increasing complexity (evaluated by the number of symbols making up the formula). If the first sequence contains all true, and the second contains all false formulas of predicate calculsu (which is not impossble in the case of the "nonconstructive medium") then it is not difficult to construct a completely constructive decision procedure for the (restricted) predicate calculus, based on the accumulation of an ever greater and greater quantity of external information and comparison of it with the formulas which are to be resolved.

It is obvious that the "nonconstructive medium" is not at all obligated to completely take upon itself the decision task, as was actually the case in the example presented. The nonconstructive sequences which it generates may not even be direct sequences of the formulas.

They must pr s only one characteristic – the possibility of their constructi\ .ansformation (within the framework of the considered self-organizing decision procedure) into suitably ordered sequences of demonstrable (true) and nondemonstrable (false) formulas of restricted predicate calculus.

It is possible that these considerations may serve in the future as the basis for systems for far-reaching automation of the processes of scientific creativity, principally the automation of the process of the construction of complex deductive theories.